

Rewriting Queries on SPARQL Views

Wangchao Le¹ Songyun Duan² Anastasios Kementsietsidis²
Feifei Li¹ Min Wang³



¹Florida State University



²IBM T.J. Watson Research Center



³HP Lab China

March 24, 2011

Outline

1 Introduction

- Problem definition
- RDF data and SPARQL language
- A running example and solution outline

2 Rewriting on SPARQL views

- Rewriting Algorithm
- Remarks on SPARQL rewriting

3 Optimization on rewritings

- Minimize a rewritten query
- Pruning rewritings with empty results

4 Experiment

5 Conclusion

Introduction

- Answering queries over *virtual* views is encountered in many settings.

Introduction

- Answering queries over *virtual* views is encountered in many settings.
 - Access control.

Introduction

- Answering queries over *virtual* views is encountered in many settings.
 - Access control.
 - Data integration.

Introduction

- Answering queries over *virtual* views is encountered in many settings.
 - Access control.
 - Data integration.
- For example, DBAs use views to enforce security clearance.

Introduction

- Answering queries over *virtual* views is encountered in many settings.
 - Access control.
 - Data integration.
- For example, DBAs use views to enforce security clearance.
- Views are materialized.

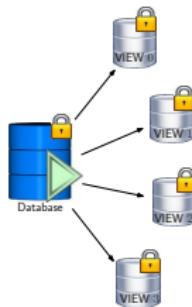
Introduction

- Answering queries over *virtual* views is encountered in many settings.
 - Access control.
 - Data integration.
- For example, DBAs use views to enforce security clearance.
- ④ Views are materialized.



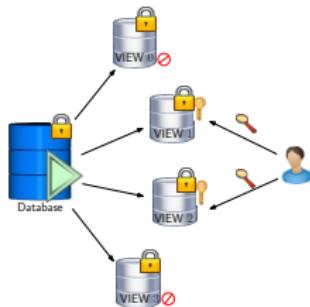
Introduction

- Answering queries over *virtual* views is encountered in many settings.
 - Access control.
 - Data integration.
- For example, DBAs use views to enforce security clearance.
- Views are materialized.



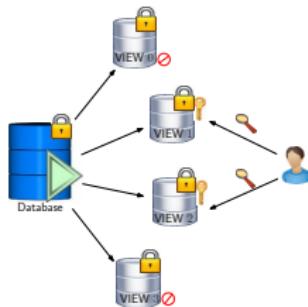
Introduction

- Answering queries over *virtual views* is encountered in many settings.
 - Access control.
 - Data integration.
- For example, DBAs use views to enforce security clearance.
- Views are materialized.



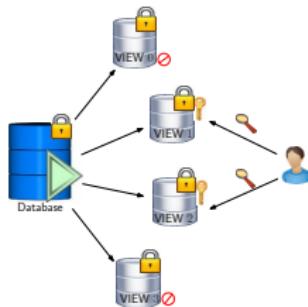
Introduction

- Answering queries over *virtual* views is encountered in many settings.
 - Access control.
 - Data integration.
 - For example, DBAs use views to enforce security clearance.
- ③ Views are materialized.
- +: Evaluation.
 - -: Space, updates, etc.



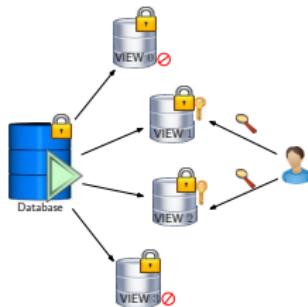
Introduction

- Answering queries over *virtual* views is encountered in many settings.
 - Access control.
 - Data integration.
 - For example, DBAs use views to enforce security clearance.
- ④ Views are materialized.
- +: Evaluation.
 - -: Space, updates, etc.



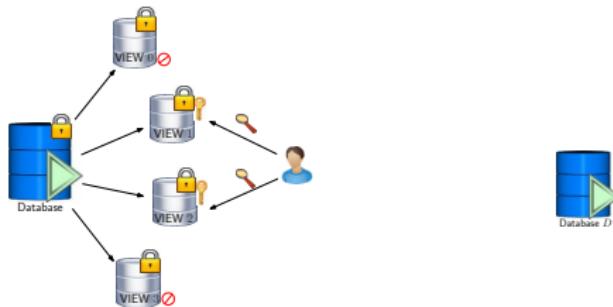
Introduction

- Answering queries over *virtual* views is encountered in many settings.
 - Access control.
 - Data integration.
 - For example, DBAs use views to enforce security clearance.
- ① Views are materialized. ② Views are virtual.
- +: Evaluation.
 - -: Space, updates, etc.



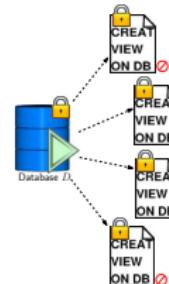
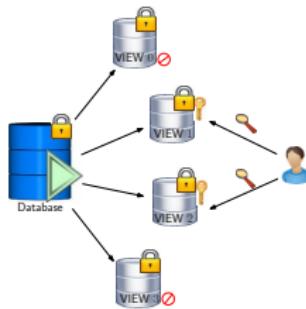
Introduction

- Answering queries over *virtual* views is encountered in many settings.
 - Access control.
 - Data integration.
 - For example, DBAs use views to enforce security clearance.
- ① Views are materialized. ② Views are virtual.
- +: Evaluation.
 - -: Space, updates, etc.



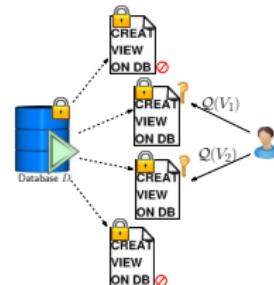
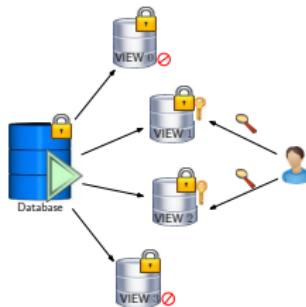
Introduction

- Answering queries over *virtual* views is encountered in many settings.
 - Access control.
 - Data integration.
 - For example, DBAs use views to enforce security clearance.
- ① Views are materialized. ② Views are virtual.
- +: Evaluation.
 - -: Space, updates, etc.



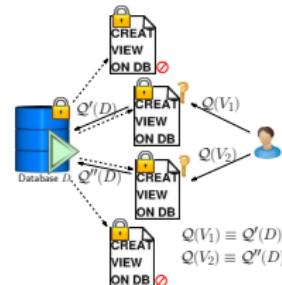
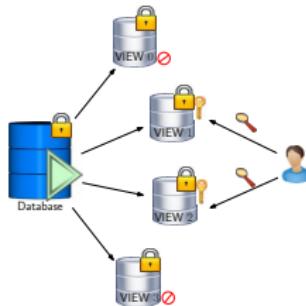
Introduction

- Answering queries over *virtual* views is encountered in many settings.
 - Access control.
 - Data integration.
 - For example, DBAs use views to enforce security clearance.
- ① Views are materialized. ② Views are virtual.
- +: Evaluation.
 - -: Space, updates, etc.



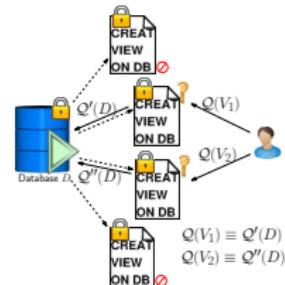
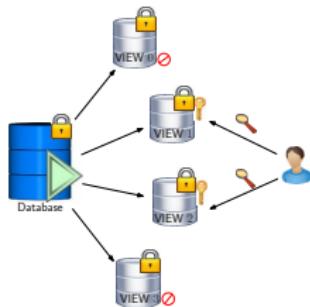
Introduction

- Answering queries over *virtual* views is encountered in many settings.
 - Access control.
 - Data integration.
 - For example, DBAs use views to enforce security clearance.
- ① Views are materialized. ② Views are virtual.
- +: Evaluation.
 - -: Space, updates, etc.



Introduction

- Answering queries over *virtual views* is encountered in many settings.
 - Access control.
 - Data integration.
 - For example, DBAs use views to enforce security clearance.
- ① Views are materialized.
 - +: Evaluation.
 - -: Space, updates, etc.
- ② Views are virtual.
 - +: Maintenance, e.g., ACID.
 - -: Rewriting is NP-hard.



Outline

1 Introduction

- Problem definition
- RDF data and SPARQL language
- A running example and solution outline

2 Rewriting on SPARQL views

- Rewriting Algorithm
- Remarks on SPARQL rewriting

3 Optimization on rewritings

- Minimize a rewritten query
- Pruning rewritings with empty results

4 Experiment

5 Conclusion

Introduction: SPARQL query rewriting

- Answering query using views is well studied for relational DBs, but not well understood in RDF and SPARQL.
- Given a set **definitions of views** $\mathcal{V} = \{V_1, V_2, \dots, V_I\}$ on a RDF graph G and SPARQL query Q over the views \mathcal{V} , compute a SPARQL query Q' such that $Q'(G) = Q(V(G))$.

Introduction: SPARQL query rewriting

- Answering query using views is well studied for relational DBs, but not well understood in RDF and SPARQL.
- Given a set **definitions of views** $\mathcal{V} = \{V_1, V_2, \dots, V_I\}$ on a RDF graph G and SPARQL query Q over the views \mathcal{V} , compute a SPARQL query Q' such that $Q'(G) = Q(V(G))$.
- **Soundness:** The rewriting is sound iff $Q'(G) \subset Q(V(G))$.

Introduction: SPARQL query rewriting

- Answering query using views is well studied for relational DBs, but not well understood in RDF and SPARQL.
- Given a set **definitions of views** $\mathcal{V} = \{V_1, V_2, \dots, V_l\}$ on a RDF graph G and SPARQL query Q over the views \mathcal{V} , compute a SPARQL query Q' such that $Q'(G) = Q(V(G))$.
- **Soundness:** The rewriting is sound iff $Q'(G) \subset Q(V(G))$.
- **Completeness:** The rewriting is complete iff $Q(V(G)) \subset Q'(G)$.

Outline

1 Introduction

- Problem definition
- RDF data and SPARQL language
- A running example and solution outline

2 Rewriting on SPARQL views

- Rewriting Algorithm
- Remarks on SPARQL rewriting

3 Optimization on rewritings

- Minimize a rewritten query
- Pruning rewritings with empty results

4 Experiment

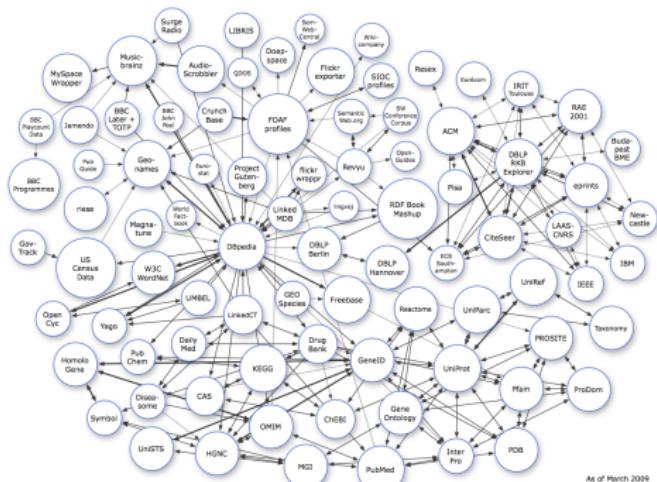
5 Conclusion

Introduction: RDF data and SPARQL language

- Resource Description Framework (RDF) is commonly used in semantic web for knowledge representation.

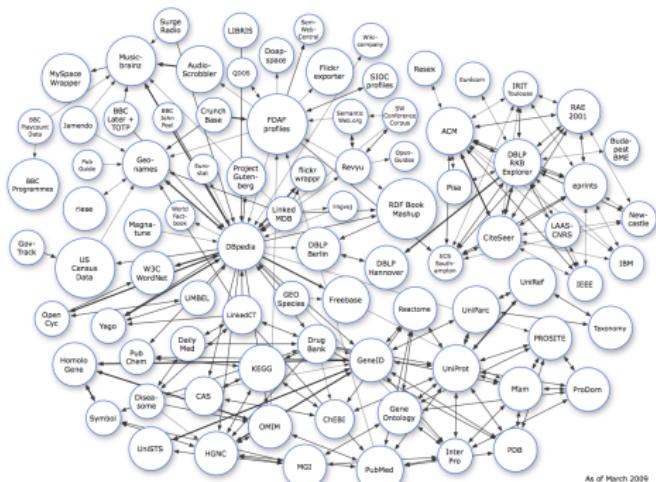
Introduction: RDF data and SPARQL language

- Resource Description Framework (RDF) is commonly used in semantic web for knowledge representation.

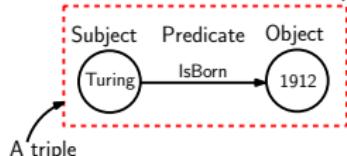


Introduction: RDF data and SPARQL language

- Resource Description Framework (RDF) is commonly used in semantic web for knowledge representation.



- DBpedia extracted unstructured information and published in queryable structured format (RDF).
 - Describing 3.5 million things.
 - Unit of RDF data: linked triple.



Introduction: RDF data and SPARQL language

- We are interested in SPARQL queries of the form
$$Q := (\text{SELECT} | \text{CONSTRUCT}) \quad RD \quad (\text{WHERE} \quad GP)$$

Introduction: RDF data and SPARQL language

- We are interested in SPARQL queries of the form
$$Q := (\text{SELECT} | \text{CONSTRUCT}) \quad RD \quad (\text{WHERE} \quad GP)$$
- Given an RDF graph G and a graph pattern GP , Q searches for subgraphs in G that match GP .

Introduction: RDF data and SPARQL language

- We are interested in SPARQL queries of the form
$$Q := (\text{SELECT} | \text{CONSTRUCT}) \quad RD \quad (\text{WHERE} \quad GP)$$
- Given an RDF graph G and a graph pattern GP , Q searches for subgraphs in G that match GP .
- RD is the result description.

Introduction: RDF data and SPARQL language

- We are interested in SPARQL queries of the form
$$Q := (\text{SELECT} | \text{CONSTRUCT}) \quad RD \quad (\text{WHERE} \quad GP)$$
- Given an RDF graph G and a graph pattern GP , Q searches for subgraphs in G that match GP .
- RD is the result description.
 - For SELECT , it projects the values of variables (like SQL).
 - For CONSTRUCT , it is a set of triple templates that constructs a new RDF graph from the matched subgraphs in G .

Introduction: RDF data and SPARQL language

- We are interested in SPARQL queries of the form
$$Q := (\text{SELECT} | \text{CONSTRUCT}) \quad RD \quad (\text{WHERE} \quad GP)$$
- Given an RDF graph G and a graph pattern GP , Q searches for subgraphs in G that match GP .
- RD is the result description.
 - For SELECT , it projects the values of variables (like SQL).
 - For CONSTRUCT , it is a set of triple templates that constructs a new RDF graph from the matched subgraphs in G .
- Finally, there is a Boolean SPARQL query of form: $\text{ASK } GP$, which indicates if GP exists, or not, in G .

Outline

1 Introduction

- Problem definition
- RDF data and SPARQL language
- A running example and solution outline

2 Rewriting on SPARQL views

- Rewriting Algorithm
- Remarks on SPARQL rewriting

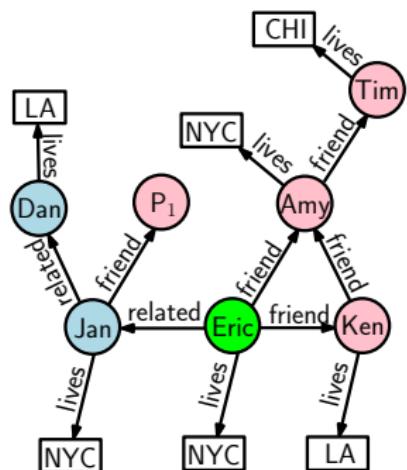
3 Optimization on rewritings

- Minimize a rewritten query
- Pruning rewritings with empty results

4 Experiment

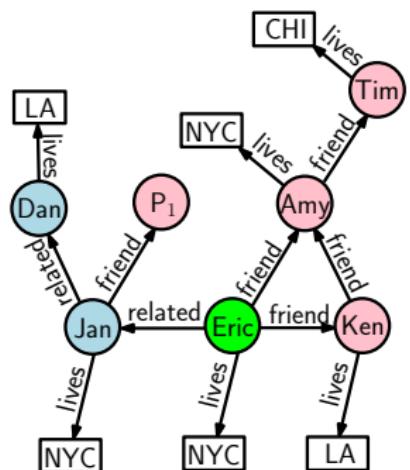
5 Conclusion

Introduction: A running example and solution outline



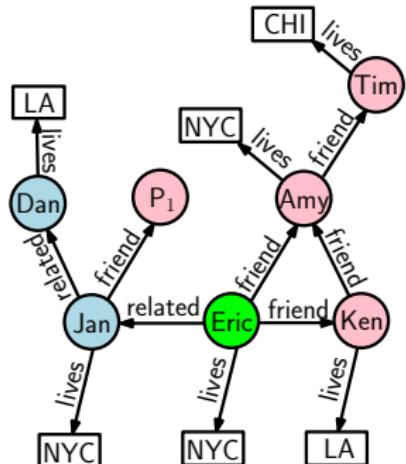
Introduction: A running example and solution outline

- In SPARQL, define view by CONSTRUCT.



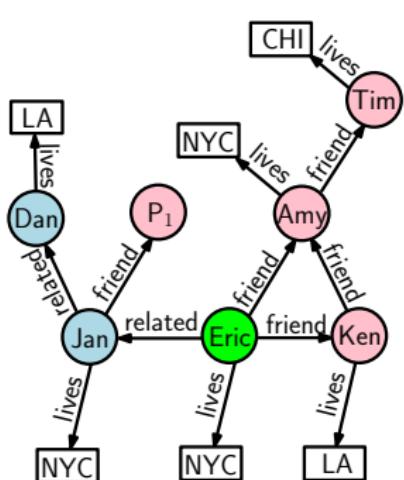
Introduction: A running example and solution outline

- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).



Introduction: A running example and solution outline

- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).

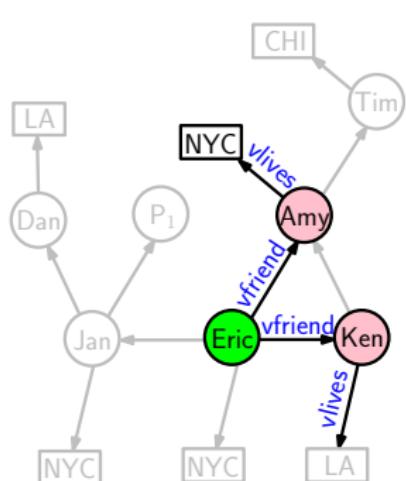


V_F :

```
CONSTRUCT{  
?f0 vfriend ?f1,  
?f1 vives ?l1}  
WHERE{  
?f0 name Eric,  
?f0 friend ?f1,  
?f1 lives ?l1}
```

Introduction: A running example and solution outline

- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).

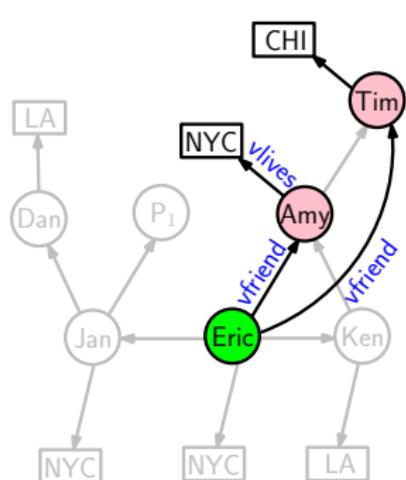


V_F :

```
CONSTRUCT{  
?f0 vfriend ?f1,  
?f1 vives ?l1}  
WHERE{  
?f0 name Eric,  
?f0 friend ?f1,  
?f1 lives ?l1}
```

Introduction: A running example and solution outline

- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).



V_F :

```
CONSTRUCT{  
?f0 vfriend ?f1,  
?f1 vlives ?l1}
```

WHERE{
?f₀ name Eric,
?f₀ friend ?f₁,
?f₁ lives ?l₁}

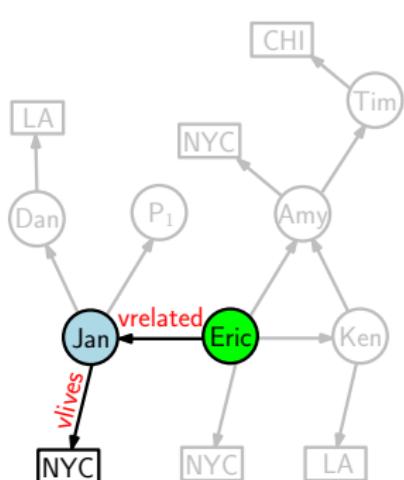
V_{FoF} :

```
CONSTRUCT{  
?f2 vfriend ?f4,  
?f4 vlives ?l4}
```

WHERE{
?f₂ name Eric,
?f₂ friend ?f₃,
?f₃ friend ?f₄,
?f₄ lives ?l₄}

Introduction: A running example and solution outline

- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).



V_F :

```
CONSTRUCT{  
?f0 vfriend ?f1,  
?f1 vlives ?l1}
```

WHERE{
?f₀ name Eric,
?f₀ friend ?f₁,
?f₁ lives ?l₁}

V_{FoF} :

```
CONSTRUCT{  
?f2 vfriend ?f4,  
?f4 vlives ?l4}
```

WHERE{
?f₂ name Eric,
?f₂ friend ?f₃,
?f₃ friend ?f₄,
?f₄ lives ?l₄}

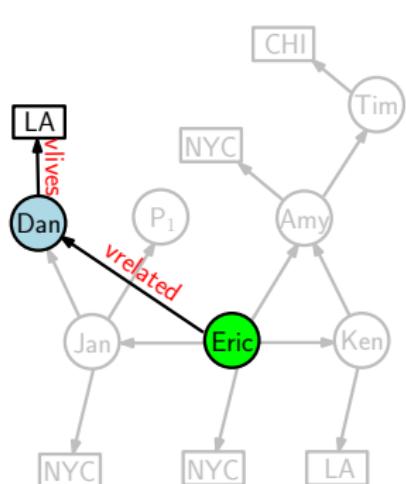
V_R :

```
CONSTRUCT{  
?r0 vrelated ?r1,  
?r1 vlives ?l1}
```

WHERE{
?r₀ name Eric,
?r₀ related ?r₁,
?r₁ lives ?l₁}

Introduction: A running example and solution outline

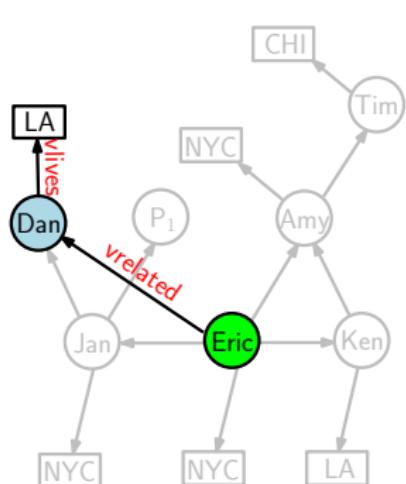
- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).



$V_F:$ CONSTRUCT{ ?f ₀ vfriend ?f ₁ , ?f ₁ vlives ?l ₁ }	$V_{FoF}:$ CONSTRUCT{ ?f ₂ vfriend ?f ₄ , ?f ₄ vlives ?l ₄ }	$V_R:$ CONSTRUCT{ ?r ₀ vrelated ?r ₁ , ?r ₁ vlives ?l ₁ }	$V_{RoR}:$ CONSTRUCT{ ?r ₂ vrelated ?r ₄ , ?r ₄ vlives ?l ₄ }
WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₁ }	WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₄ }

Introduction: A running example and solution outline

- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).



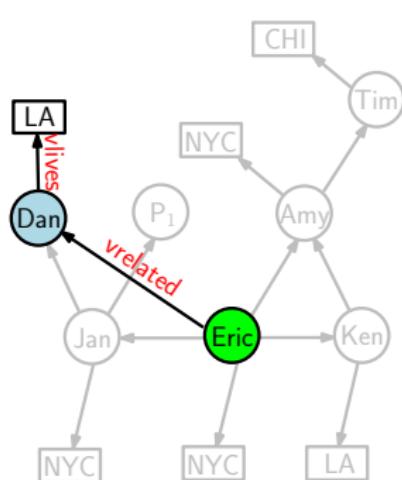
$V_F:$ CONSTRUCT{ $?f_0 \text{ vfriend } ?f_1,$ $?f_1 \text{ vlives } ?l_1\}$	$V_{FoF}:$ CONSTRUCT{ $?f_2 \text{ vfriend } ?f_4,$ $?f_4 \text{ vlives } ?l_4\}$	$V_R:$ CONSTRUCT{ $?r_0 \text{ vrelated } ?r_1,$ $?r_1 \text{ vlives } ?l_1\}$	$V_{RoR}:$ CONSTRUCT{ $?r_2 \text{ vrelated } ?r_4,$ $?r_4 \text{ vlives } ?l_4\}$
WHERE{ $?f_0 \text{ name Eric},$ $?f_0 \text{ friend } ?f_1,$ $?f_1 \text{ lives } ?l_1\}$	WHERE{ $?f_2 \text{ name Eric},$ $?f_2 \text{ friend } ?f_3,$ $?f_3 \text{ friend } ?f_4,$ $?f_4 \text{ lives } ?l_4\}$	WHERE{ $?r_0 \text{ name Eric},$ $?r_0 \text{ related } ?r_1,$ $?r_1 \text{ lives } ?l_1\}$	WHERE{ $?r_2 \text{ name Eric},$ $?r_2 \text{ related } ?r_3,$ $?r_3 \text{ related } ?r_4,$ $?r_4 \text{ lives } ?l_4\}$

$Q_u:$

```
SELECT ?f5
WHERE {?p vfriend ?f5, ?f5 vlives ?l5,
       ?p vrelated ?r5, ?r5 vlives ?l5.}
```

Introduction: A running example and solution outline

- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).



$V_F:$
CONSTRUCT{
?f₀ vfriend ?f₁,
?f₁ vlives ?l₁}

WHERE{
?f₀ name Eric,
?f₀ friend ?f₁,
?f₁ lives ?l₁}

$V_{FoF}:$
CONSTRUCT{
?f₂ vfriend ?f₄,
?f₄ vlives ?l₄}

WHERE{
?f₂ name Eric,
?f₂ friend ?f₃,
?f₃ friend ?f₄,
?f₄ lives ?l₄}

$V_R:$
CONSTRUCT{
?r₀ vrelated ?r₁,
?r₁ vlives ?l₁}

WHERE{
?r₀ name Eric,
?r₀ related ?r₁,
?r₁ lives ?l₁}

$V_{RoR}:$
CONSTRUCT{
?r₂ vrelated ?r₄,
?r₄ vlives ?l₄}

WHERE{
?r₂ name Eric,
?r₂ related ?r₃,
?r₃ related ?r₄,
?r₄ lives ?l₄}

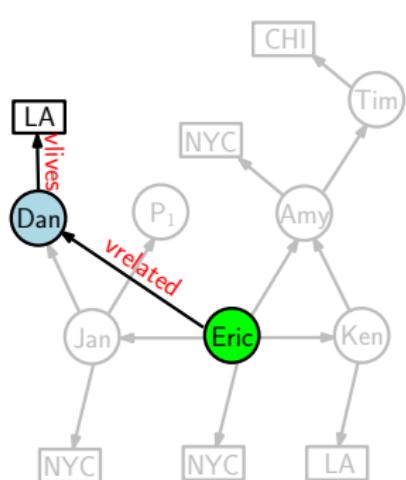
$Q_u:$

SELECT ?f₅

WHERE {?p vfriend ?f₅ , ?f₅ vlives ?l₅,
?p vrelated ?r₅ , ?r₅ vlives ?l₅.}

Introduction: A running example and solution outline

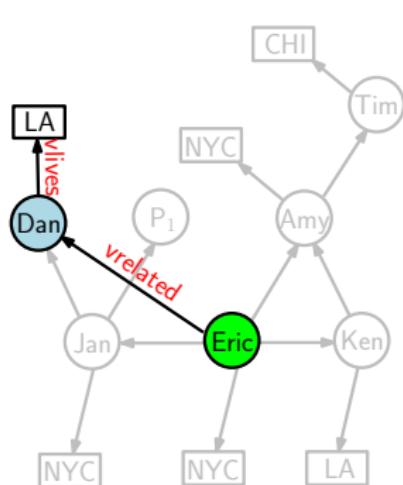
- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).



$V_F:$ CONSTRUCT{ $?f_0 \text{ vfriend } ?f_1,$ $?f_1 \text{ vlivels } ?l_1\}$	$V_{FoF}:$ CONSTRUCT{ $?f_2 \text{ vfriend } ?f_4,$ $?f_4 \text{ vlivels } ?l_4\}$	$V_R:$ CONSTRUCT{ $?r_0 \text{ vrelated } ?r_1,$ $?r_1 \text{ vlivels } ?l_1\}$	$V_{RoR}:$ CONSTRUCT{ $?r_2 \text{ vrelated } ?r_4,$ $?r_4 \text{ vlivels } ?l_4\}$
WHERE{ $?f_0 \text{ name Eric},$ $?f_0 \text{ friend } ?f_1,$ $?f_1 \text{ lives } ?l_1\}$	WHERE{ $?f_2 \text{ name Eric},$ $?f_2 \text{ friend } ?f_3,$ $?f_3 \text{ friend } ?f_4,$ $?f_4 \text{ lives } ?l_4\}$	WHERE{ $?r_0 \text{ name Eric},$ $?r_0 \text{ related } ?r_1,$ $?r_1 \text{ lives } ?l_1\}$	WHERE{ $?r_2 \text{ name Eric},$ $?r_2 \text{ related } ?r_3,$ $?r_3 \text{ related } ?r_4,$ $?r_4 \text{ lives } ?l_4\}$
Q _u : SELECT ?f _n WHERE {?p vfriend ?f ₅ , ?f ₅ vlivels ?l ₅ , ?p vrelated ?r ₅ , ?r ₅ vlivels ?l ₅ .}			

Introduction: A running example and solution outline

- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).



$V_F:$
CONSTRUCT{
?f₀ **vfriend** ?f₁
?f₁ vlives ?l₁}
WHERE{
?f₀ name Eric,
?f₀ friend ?f₁,
?f₁ lives ?l₁}

$V_{FoF}:$
CONSTRUCT{
?f₂ **vfriend** ?f₄
?f₄ vlives ?l₄}
WHERE{
?f₂ name Eric,
?f₂ friend ?f₃,
?f₃ friend ?f₄,
?f₄ lives ?l₄}

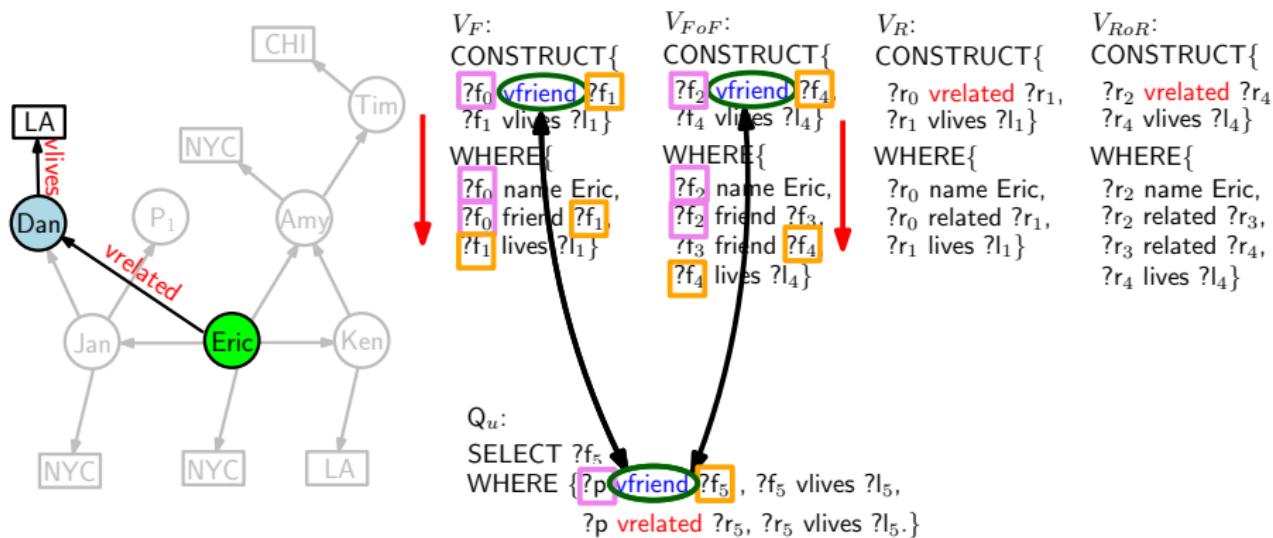
$V_R:$
CONSTRUCT{
?r₀ **vrelated** ?r₁,
?r₁ vlives ?l₁}
WHERE{
?r₀ name Eric,
?r₀ related ?r₁,
?r₁ lives ?l₁}

$V_{RoR}:$
CONSTRUCT{
?r₂ **vrelated** ?r₄,
?r₄ vlives ?l₄}
WHERE{
?r₂ name Eric,
?r₂ related ?r₃,
?r₃ related ?r₄,
?r₄ lives ?l₄}

$Q_u:$
SELECT ?f₅
WHERE {?p **vfriend** ?f₅, ?f₅ vlives ?l₅,
?p **vrelated** ?r₅, ?r₅ vlives ?l₅.}

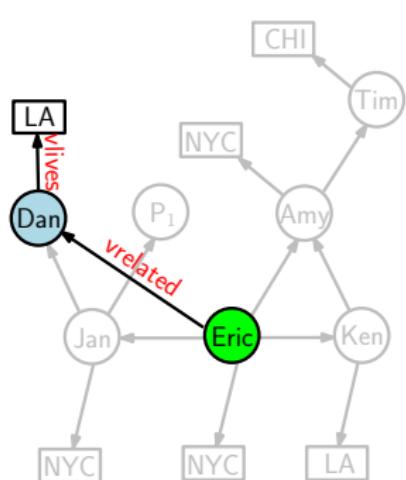
Introduction: A running example and solution outline

- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).



Introduction: A running example and solution outline

- In SPARQL, define view by CONSTRUCT.
- Eric wants to hide his RoF, FoR and any distinction between FoF(RoR) and F(R).



$V_F:$
CONSTRUCT{
?f₀ **vfriend** ?f₁,
?f₁ **vlives** ?l₁}

WHERE{
?f₀ name Eric,
?f₀ friend ?f₁,
?f₁ lives ?l₁}

$V_{FoF}:$
CONSTRUCT{
?f₂ **vfriend** ?f₄,
?f₄ **vlives** ?l₄}

WHERE{
?f₂ name Eric,
?f₂ friend ?f₃,
?f₃ friend ?f₄,
?f₄ lives ?l₄}

$V_R:$
CONSTRUCT{
?r₀ **vrelated** ?r₁,
?r₁ **vlives** ?l₁}

WHERE{
?r₀ name Eric,
?r₀ related ?r₁,
?r₁ lives ?l₁}

$V_{RoR}:$
CONSTRUCT{
?r₂ **vrelated** ?r₄,
?r₄ **vlives** ?l₄}

WHERE{
?r₂ name Eric,
?r₂ related ?r₃,
?r₃ related ?r₄,
?r₄ lives ?l₄}

$Q_u:$

SELECT ?f₅
WHERE {?p **vfriend** ?f₅, ?f₅ **vlives** ?l₅,
?p **vrelated** ?r₅, ?r₅ **vlives** ?l₅.}

Output exponential rewritings!

Outline

1 Introduction

- Problem definition
- RDF data and SPARQL language
- A running example and solution outline

2 Rewriting on SPARQL views

- Rewriting Algorithm
- Remarks on SPARQL rewriting

3 Optimization on rewritings

- Minimize a rewritten query
- Pruning rewritings with empty results

4 Experiment

5 Conclusion

Outline

1 Introduction

- Problem definition
- RDF data and SPARQL language
- A running example and solution outline

2 Rewriting on SPARQL views

- **Rewriting Algorithm**
- Remarks on SPARQL rewriting

3 Optimization on rewritings

- Minimize a rewritten query
- Pruning rewritings with empty results

4 Experiment

5 Conclusion

Rewriting Algorithm

$Q_u:$
SELECT ?f₅ WHERE { ?p vfriend ?f₅ ?f₅ vives ?l₅ ?p vrelated ?r₅ ?r₅ vives ?l₅ }

$V_F:$ CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ , ② ?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	$V_{FoF}:$ CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ , ② ?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	$V_R:$ CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ , ② ?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ }	$V_{RoR}:$ CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ , ② ?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }
---	---	---	--

Rewriting Algorithm

Building CandV₁



Q_u:

SELECT ?f₅ WHERE {

CandV ₁	CandV ₂	CandV ₃	CandV ₄
?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅

}

V _F : CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ , ② ?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	V _{FoF} : CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ , ② ?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	V _R : CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ , ② ?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ }	V _{RoR} : CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ , ② ?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }
---	---	---	--

Rewriting Algorithm

Building CandV₁



Q _u :	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅

[V_F^①, ?f₀ → ?p, ?f₁ → ?f₅]

V _F :	V _{FoF} :	V _R :	V _{RoR} :
CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ ② ?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ , ② ?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ , ② ?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ }	CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ , ② ?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }

Rewriting Algorithm

Building CandV₁



Q _u :	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅

[V_F^①, ?f₀ → ?p, ?f₁ → ?f₅]
[V_{FoF}^②, ?f₂ → ?p, ?f₄ → ?f₅]

V _F :	V _{FoF} :	V _R :	V _{RoR} :
CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ , ② ?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ , ② ?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ , ② ?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ }	CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ , ② ?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }

Rewriting Algorithm

Building CandV₂

Q_u:
SELECT ?f₅ WHERE {

CandV ₁	CandV ₂	CandV ₃	CandV ₄
?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅

}

[V_F⁰,?f₀→?p,?f₁→?f₅] [V_F⁰,?f₁→?f₅,?l₁→?l₅]

[V_{FoF}⁰,?f₂→?p,?f₄→?f₅] [V_{FoF}⁰,?f₄→?f₅,?l₄→?l₅]

[V_R⁰,?r₁→?f₅,?l₃→?l₅]

[V_{RoR}⁰,?r₄→?f₅,?l₆→?l₅]

V_F:
CONSTRUCT{
① ?f₀ vfriend ?f₁,
② ?f₁ vives ?l₁} WHERE{
?f₀ name Eric,
?f₀ friend ?f₁,
?f₁ lives ?l₁}

V_{FoF}:
CONSTRUCT{
① ?f₂ vfriend ?f₄,
② ?f₄ vives ?l₄} WHERE{
?f₂ name Eric,
?f₂ friend ?f₃,
?f₃ friend ?f₄,
?f₄ lives ?l₄}

V_R:
CONSTRUCT{
① ?r₀ vrelated ?r₁,
② ?r₁ vives ?l₃} WHERE{
?r₀ name Eric,
?r₀ related ?r₁,
?r₁ lives ?l₃}

V_{RoR}:
CONSTRUCT{
① ?r₂ vrelated ?r₄,
② ?r₄ vives ?l₆} WHERE{
?r₂ name Eric,
?r₂ related ?r₃,
?r₃ related ?r₄,
?r₄ lives ?l₆}

Rewriting Algorithm

Building CandV₃

↓

Q _u :	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ ,?f ₀ →?p,?f ₁ →?f ₅]	[V _F ⁰ ,?f ₁ →?f ₅ ,?l ₁ →?l ₅]	[V _R ⁰ ,?r ₀ →?p,?r ₁ →?r ₅]	
	[V _{FoF} ⁰ ,?f ₂ →?p,?f ₄ →?f ₅]	[V _{FoF} ⁰ ,?f ₄ →?f ₅ ,?l ₄ →?l ₅]	[V _{RoR} ⁰ ,?r ₂ →?p,?r ₄ →?r ₅]	
	[V _R ⁰ ,?r ₁ →?f ₅ ,?l ₃ →?l ₅]			
	[V _{RoR} ⁰ ,?r ₄ →?f ₅ ,?l ₆ →?l ₅]			

V _F :	V _{FoF} :	V _R :	V _{RoR} :
CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ , ② ?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ , ② ?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ , ② ?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ }	CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ , ② ?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }
		① ?r ₀ vrelated ?r ₁ , ② ?r ₁ vives ?l ₃	① ?r ₂ vrelated ?r ₄ , ② ?r ₄ vives ?l ₆

Rewriting Algorithm

Building CandV₄

↓

Q _u :	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅] [V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅] [V _R ⁰ , ?r ₀ → ?p, ?r ₁ → ?r ₅] [V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]			
	[V _{FoF} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅] [V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅] [V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅] [V _{FoF} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]	[V _R ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]		[V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅]
		[V _{RoR} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]		[V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]

$V_F:$ CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ , ② ?f₁ vives ?l₁ WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ } 	$V_{FoF}:$ CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ , ② ?f₄ vives ?l₄ WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ } 	$V_R:$ CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ , ② ?r₁ vives ?l₃ WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ } 	$V_{RoR}:$ CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ , ② ?r₄ vives ?l₆ WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }
---	---	---	--

Rewriting Algorithm

$Q_u:$	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅] [V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅] [V _R ⁰ , r ₀ → ?p, ?r ₁ → ?r ₅] [V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]			
	[V _{FoF} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅] [V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅] [V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅] [V _{FoF} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]	[V _R ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]		[V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅]
		[V _{FoR} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]		[V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]

$V_F:$	$V_{FoF}:$	$V_R:$	$V_{RoR}:$
CONSTRUCT{	CONSTRUCT{	CONSTRUCT{	CONSTRUCT{
① ?f ₀ vfriend ?f ₁ ,	① ?f ₂ vfriend ?f ₄ ,	① ?r ₀ vrelated ?r ₁ ,	① ?r ₂ vrelated ?r ₄ ,
② ?f ₁ vives ?l ₁ }	② ?f ₄ vives ?l ₄ }	② ?r ₁ vives ?l ₃ }	② ?r ₄ vives ?l ₆ }
WHERE{	WHERE{	WHERE{	WHERE{
?f ₀ name Eric,	?f ₂ name Eric,	?r ₀ name Eric,	?r ₂ name Eric,
?f ₀ friend ?f ₁ ,	?f ₂ friend ?f ₃ ,	?r ₀ related ?r ₁ ,	?r ₂ related ?r ₃ ,
?f ₁ lives ?l ₁ }	?f ₃ friend ?f ₄ ,	?r ₁ lives ?l ₃ }	?r ₃ related ?r ₄ ,
	?f ₄ lives ?l ₄ }		?r ₄ lives ?l ₆ }

Rewriting Algorithm

Generate one rewriting:

$Q_u:$	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅] [V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅] [V _R ⁰ , r ₀ → ?p, ?r ₁ → ?r ₅] [V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]			
	[V _{FoF} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅] [V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅] [V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅] [V _{FoF} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]	[V _R ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]		[V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅]
		[V _{FoR} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]		[V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]

$V_F:$ CONSTRUCT{	$V_{FoF}:$ CONSTRUCT{	$V_R:$ CONSTRUCT{	$V_{RoR}:$ CONSTRUCT{
① ?f ₀ vfriend ?f ₁ , ② ?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	① ?f ₂ vfriend ?f ₄ , ② ?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	① ?r ₀ vrelated ?r ₁ , ② ?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ }	① ?r ₂ vrelated ?r ₄ , ② ?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }

Rewriting Algorithm

Generate one rewriting:

SELECT ?f₅ WHERE {

Q _u :	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ ,?f ₀ →?p,?f ₁ →?f ₅]	[V _F ⁰ ,?f ₁ →?f ₅ ,?l ₁ →?l ₅]	[V _R ⁰ ,r ₀ →?p,?r ₁ →?r ₅]	[V _F ⁰ ,?f ₁ →?r ₅ ,?l ₁ →?l ₅]
	[V _{FoF} ⁰ ,?f ₂ →?p,?f ₄ →?f ₅]	[V _{FoF} ⁰ ,?f ₄ →?f ₅ ,?l ₄ →?l ₅]	[V _{RoR} ⁰ ,?r ₂ →?p,?r ₄ →?r ₅]	[V _{FoF} ⁰ ,?f ₄ →?r ₅ ,?l ₄ →?l ₅]
		[V _R ⁰ ,?r ₁ →?f ₅ ,?l ₃ →?l ₅]		[V _R ⁰ ,?r ₁ →?r ₅ ,?l ₃ →?l ₅]
		[V _{RoR} ⁰ ,?r ₄ →?f ₅ ,?l ₆ →?l ₅]		[V _{RoR} ⁰ ,?r ₄ →?r ₅ ,?l ₆ →?l ₅]

V_F:

CONSTRUCT{
① ?f₀ vfriend ?f₁,
② ?f₁ vives ?l₁}

WHERE{
?f₀ name Eric,
?f₀ friend ?f₁,
?f₁ lives ?l₁}

V_{FoF}:

CONSTRUCT{
① ?f₂ vfriend ?f₄,
② ?f₄ vives ?l₄}

WHERE{
?f₂ name Eric,
?f₂ friend ?f₃,
?f₃ friend ?f₄,
?f₄ lives ?l₄}

V_R:

CONSTRUCT{
① ?r₀ vrelated ?r₁,
② ?r₁ vives ?l₃}

WHERE{
?r₀ name Eric,
?r₀ related ?r₁,
?r₁ lives ?l₃}

V_{RoR}:

CONSTRUCT{
① ?r₂ vrelated ?r₄,
② ?r₄ vives ?l₆}

WHERE{
?r₂ name Eric,
?r₂ related ?r₃,
?r₃ related ?r₄,
?r₄ lives ?l₆}

Rewriting Algorithm

Generate one rewriting:

SELECT ?f₅ WHERE {

Q _u :	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ ,?f ₀ →?p,?f ₁ →?f ₅]	[V _F ⁰ ,?f ₁ →?f ₅ ,?l ₁ →?l ₅]	[V _R ⁰ ,r ₀ →?p,?r ₁ →?r ₅]	[V _F ⁰ ,?f ₁ →?r ₅ ,?l ₁ →?l ₅]
	[V _{FoF} ⁰ ,?f ₂ →?p,?f ₄ →?f ₅]	[V _{FoF} ⁰ ,?f ₄ →?f ₅ ,?l ₄ →?l ₅]	[V _{RoR} ⁰ ,?r ₂ →?p,?r ₄ →?r ₅]	[V _{FoF} ⁰ ,?f ₄ →?r ₅ ,?l ₄ →?l ₅]
		[V _R ⁰ ,?r ₁ →?f ₅ ,?l ₃ →?l ₅]		[V _R ⁰ ,?r ₁ →?r ₅ ,?l ₃ →?l ₅]
		[V _{RoR} ⁰ ,?r ₄ →?f ₅ ,?l ₆ →?l ₅]		[V _{RoR} ⁰ ,?r ₄ →?r ₅ ,?l ₆ →?l ₅]
<hr/>				
V _F :	V _{FoF} :	V _R :	V _{RoR} :	
CONSTRUCT{	CONSTRUCT{	CONSTRUCT{	CONSTRUCT{	
① ?f ₀ vfriend ?f ₁ ,	① ?f ₂ vfriend ?f ₄ ,	① ?r ₀ vrelated ?r ₁ ,	① ?r ₂ vrelated ?r ₄ ,	
② ?f ₁ vives ?l ₁ }	② ?f ₄ vives ?l ₄ }	② ?r ₁ vives ?l ₃ }	② ?r ₄ vives ?l ₆ }	
WHERE{	WHERE{	WHERE{	WHERE{	
?f ₀ name Eric,	?f ₂ name Eric,	?r ₀ name Eric,	?r ₂ name Eric,	
?f ₀ friend ?f ₁ ,	?f ₂ friend ?f ₃ ,	?r ₀ related ?r ₁ ,	?r ₂ related ?r ₃ ,	
?f ₁ lives ?l ₁ }	?f ₃ friend ?f ₄ ,	?r ₁ lives ?l ₃ }	?r ₃ related ?r ₄ ,	
	?f ₄ lives ?l ₄ }		?r ₄ lives ?l ₆ }	

Rewriting Algorithm

Generate one rewriting:

SELECT ?f₅ WHERE { ?p name Eric, ?p friend ?f₅, ?f₅ lives ?l₁,

Q _u :	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ ,?f ₀ →?p,?f ₁ →?f ₅]	[V _F ⁰ ,?f ₁ →?f ₅ ,?l ₁ →?l ₅]	[V _R ⁰ ,?r ₀ →?p,?r ₁ →?r ₅]	[V _F ⁰ ,?f ₁ →?r ₅ ,?l ₁ →?l ₅]
	[V _{FoF} ⁰ ,?f ₂ →?p,?f ₄ →?f ₅]	[V _{FoF} ⁰ ,?f ₄ →?f ₅ ,?l ₄ →?l ₅]	[V _{RoR} ⁰ ,?r ₂ →?p,?r ₄ →?r ₅]	[V _{FoF} ⁰ ,?f ₄ →?r ₅ ,?l ₄ →?l ₅]
		[V _R ⁰ ,?r ₁ →?f ₅ ,?l ₃ →?l ₅]		[V _R ⁰ ,?r ₁ →?r ₅ ,?l ₃ →?l ₅]
		[V _{RoR} ⁰ ,?r ₄ →?f ₅ ,?l ₆ →?l ₅]		[V _{RoR} ⁰ ,?r ₄ →?r ₅ ,?l ₆ →?l ₅]
<hr/>				
V _F :	V _{FoF} :	V _R :	V _{RoR} :	
CONSTRUCT{	CONSTRUCT{	CONSTRUCT{	CONSTRUCT{	
① ?f ₀ vfriend ?f ₁ ,	① ?f ₂ vfriend ?f ₄ ,	① ?r ₀ vrelated ?r ₁ ,	① ?r ₂ vrelated ?r ₄ ,	
② ?f ₁ vives ?l ₁ }	② ?f ₄ vives ?l ₄ }	② ?r ₁ vives ?l ₃ }	② ?r ₄ vives ?l ₆ }	
WHERE{	WHERE{	WHERE{	WHERE{	
?f ₀ name Eric,	?f ₂ name Eric,	?r ₀ name Eric,	?r ₂ name Eric,	
?f ₀ friend ?f ₁ ,	?f ₂ friend ?f ₃ ,	?r ₀ related ?r ₁ ,	?r ₂ related ?r ₃ ,	
?f ₁ lives ?l ₁ }	?f ₃ friend ?f ₄ ,	?r ₁ lives ?l ₃ }	?r ₃ related ?r ₄ ,	
	?f ₄ lives ?l ₄ }		?r ₄ lives ?l ₆ }	

Rewriting Algorithm

Generate one rewriting:

SELECT ?f₅ WHERE { ?p name Eric, ?p friend ?f₅, ?f₅ lives ?l' }₁

Q _u :	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅] [V _{FoF} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅] [V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅] [V _{RoR} ⁰ , ?r ₄ → ?p, ?r ₂ → ?r ₅]	[V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅] [V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅] [V _R ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅] [V _{RoR} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]	[V _R ⁰ , ?r ₀ → ?p, ?r ₁ → ?r ₅] [V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅] [V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅] [V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]	[V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅] [V _{FoF} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅] [V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅] [V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]
	V _F : CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ , ② ?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	V _{FoF} : CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ , ② ?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	V _R : CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ , ② ?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ }	V _{RoR} : CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ , ② ?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }

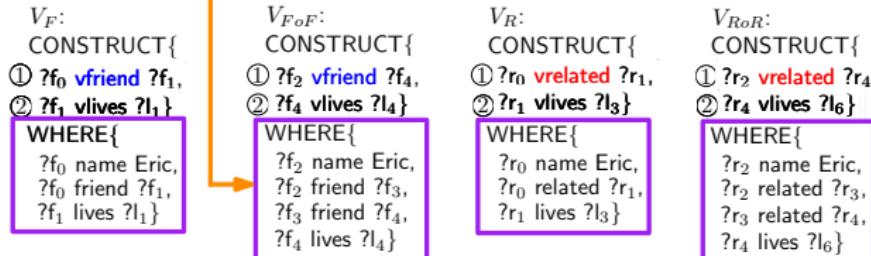
Undefined variable is renamed to a new variable.

Rewriting Algorithm

Generate one rewriting:

SELECT ?f₅ WHERE { ?p name Eric, ?p friend ?f₅, ?f₅ lives ?l₁, ?f₂ name Eric, ?f₂ friend ?f₃, ?f₃ friend ?f₅, ?f₅ lives ?l₅ }

Q _u :	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅] [V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅] [V _R ⁰ , ?r ₀ → ?p, ?r ₁ → ?r ₅] [V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]			
	[V _{FoF} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅] [V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅] [V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅] [V _{FoF} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]	[V _R ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]		[V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅]
		[V _{RoR} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]		[V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]



Rewriting Algorithm

Generate one rewriting:

SELECT ?f₅ WHERE { ?p name Eric, ?p friend ?f₅, ?f₅ lives ?l₁, ?f'₂ name Eric, ?f'₂ friend ?f'₃, ?f'₃ friend ?f₅, ?f₅ lives ?l₅
 ?p name Eric, ?p related ?r₅, ?r₅ lives ?l'₃,

Q _u :	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅] [V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅]	[V _{FoF} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅] [V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅]	[V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅] [V _{RoR} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]	[V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅] [V _{FoF} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅] [V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅] [V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]
	[V _R ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]			

V _F :	V _{FoF} :	V _R :	V _{RoR} :
CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ , ② ?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ , ② ?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ , ② ?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ }	CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ , ② ?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }

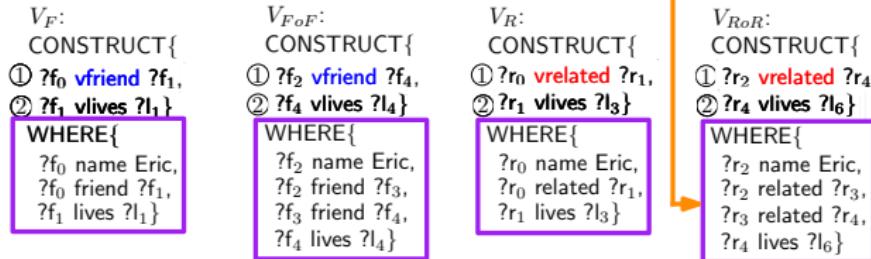
Rewriting Algorithm

Generate one rewriting:

```
SELECT ?f5 WHERE { ?p name Eric, ?p friend ?f5, ?f5 lives ?l1, ?f'2 name Eric, ?f'2 friend ?f3, ?f'3 friend ?f5, ?f5 lives ?l5  

?p name Eric, ?p related ?r5, ?r5 lives ?l'3, ?r'2 name Eric, ?r'2 related ?r3, ?r'3 related ?r5, ?r5 lives ?l'5}
```

Q _u :	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅] [V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅] [V _R ⁰ , ?r ₀ → ?p, ?r ₁ → ?r ₅] [V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]			
	[V _{FoF} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅] [V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅] [V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅] [V _{FoF} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]	[V _R ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]		[V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅]
		[V _{RoR} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]		[V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]



Rewriting Algorithm

Generate one rewriting:

SELECT ?f₅ WHERE { ?p name Eric, ?p friend ?f₅, ?f₅ lives ?l₁, ?f'₂ name Eric, ?f'₂ friend ?f₃, ?f'₃ friend ?f₅, ?f₅ lives ?l₅
?p name Eric, ?p related ?r₅, ?r₅ lives ?l'₃, ?r'₂ name Eric, ?r'₂ related ?r₃, ?r'₃ related ?r₅, ?r₅ lives ?l'₅ }

Q _u :	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅] [V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅] [V _R ⁰ , ?r ₀ → ?p, ?r ₁ → ?r ₅] [V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]			
	[V _{FoF} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅] [V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅] [V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅] [V _{FoF} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]	[V _R ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]		[V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅]
		[V _{RoR} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]		[V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]

V_F:

CONSTRUCT{
① ?f₀ vfriend ?f₁,
② ?f₁ vives ?l₁}

WHERE{
?f₀ name Eric,
?f₀ friend ?f₁,
?f₁ lives ?l₁}

V_{FoF}:

CONSTRUCT{
① ?f₂ vfriend ?f₄,
② ?f₄ vives ?l₄}

WHERE{
?f₂ name Eric,
?f₂ friend ?f₃,
?f₃ friend ?f₄,
?f₄ lives ?l₄}

V_R:

CONSTRUCT{
① ?r₀ vrelated ?r₁,
② ?r₁ vives ?l₃}

WHERE{
?r₀ name Eric,
?r₀ related ?r₁,
?r₁ lives ?l₃}

V_{RoR}:

CONSTRUCT{
① ?r₂ vrelated ?r₄,
② ?r₄ vives ?l₆}

WHERE{
?r₂ name Eric,
?r₂ related ?r₃,
?r₃ related ?r₄,
?r₄ lives ?l₆}

- There are totally |CandV₁| × |CandV₂| × |CandV₃| × |CandV₄| rewritings modulus a compatible check.

Rewriting Algorithm

Generate one rewriting:

SELECT ?f₅ WHERE { ?p name Eric, ?p friend ?f₅, ?f₅ lives ?l₁, ?f'₂ name Eric, ?f'₂ friend ?f₃, ?f'₃ friend ?f₅, ?f₅ lives ?l₅
?p name Eric, ?p related ?r₅, ?r₅ lives ?l'₃, ?r'₂ name Eric, ?r'₂ related ?r₃, ?r'₃ related ?r₅, ?r₅ lives ?l'₅ }

Q _u :	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅] [V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅] [V _R ⁰ , ?r ₀ → ?p, ?r ₁ → ?r ₅] [V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]			
	[V _{FoF} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅] [V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅] [V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅] [V _{FoF} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]	[V _R ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]	[V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅]	
		[V _{RoR} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]		[V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]

V _F :	V _{FoF} :	V _R :	V _{RoR} :
CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ , ② ?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ , ② ?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ , ② ?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ }	CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ , ② ?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }

- There are totally $|CandV_1| \times |CandV_2| \times |CandV_3| \times |CandV_4|$ rewritings modulos a compatible check.
- CandVs are considered *incompatible* if they map the same variable to different constants.

Outline

1 Introduction

- Problem definition
- RDF data and SPARQL language
- A running example and solution outline

2 Rewriting on SPARQL views

- Rewriting Algorithm
- Remarks on SPARQL rewriting

3 Optimization on rewritings

- Minimize a rewritten query
- Pruning rewritings with empty results

4 Experiment

5 Conclusion

Rewriting Algorithm: Remarks

- The rewriting algorithm could result in exponential number of rewritings.

Rewriting Algorithm: Remarks

- The rewriting algorithm could result in exponential number of rewritings.
- Can we do better? What if we translate everything to SQL and do rewriting there? Well, we proved the following result ...

Rewriting Algorithm: Remarks

- The rewriting algorithm could result in exponential number of rewritings.
- Can we do better? What if we translate everything to SQL and do rewriting there? Well, we proved the following result ...

Theorem

*The set of rewritten SPARQL queries is **sound** and **complete**.*

Rewriting Algorithm: Remarks

- The rewriting algorithm could result in exponential number of rewritings.
- Can we do better? What if we translate everything to SQL and do rewriting there? Well, we proved the following result ...

Theorem

*The set of rewritten SPARQL queries is **sound** and **complete**.*

- So, there is no redundancy nor misses resulted from the rewriting.

Rewriting Algorithm: Remarks

- The rewriting algorithm could result in exponential number of rewritings.
- Can we do better? What if we translate everything to SQL and do rewriting there? Well, we proved the following result ...

Theorem

*The set of rewritten SPARQL queries is **sound** and **complete**.*

- So, there is no redundancy nor misses resulted from the rewriting.
- So, doing rewriting in SQL will not give less rewritings as the set of semantic units is tight already.

Rewriting Algorithm: Remarks

- The rewriting algorithm could result in exponential number of rewritings.
- Can we do better? What if we translate everything to SQL and do rewriting there? Well, we proved the following result ...

Theorem

*The set of rewritten SPARQL queries is **sound** and **complete**.*

- So, there is no redundancy nor misses resulted from the rewriting.
- So, doing rewriting in SQL will not give less rewritings as the set of semantic units is tight already.
- In the experiment, we show that SQL translation is indeed helpless.

Rewriting Algorithm: Remarks

- The rewriting algorithm could result in exponential number of rewritings.
- Can we do better? What if we translate everything to SQL and do rewriting there? Well, we proved the following result ...

Theorem

*The set of rewritten SPARQL queries is **sound** and **complete**.*

- So, there is no redundancy nor misses resulted from the rewriting.
- So, doing rewriting in SQL will not give less rewritings as the set of semantic units is tight already.
- In the experiment, we show that SQL translation is indeed helpless.
- But still, can we do better?

Outline

1 Introduction

- Problem definition
- RDF data and SPARQL language
- A running example and solution outline

2 Rewriting on SPARQL views

- Rewriting Algorithm
- Remarks on SPARQL rewriting

3 Optimization on rewritings

- Minimize a rewritten query
- Pruning rewritings with empty results

4 Experiment

5 Conclusion

Outline

1 Introduction

- Problem definition
- RDF data and SPARQL language
- A running example and solution outline

2 Rewriting on SPARQL views

- Rewriting Algorithm
- Remarks on SPARQL rewriting

3 Optimization on rewritings

- Minimize a rewritten query
- Pruning rewritings with empty results

4 Experiment

5 Conclusion

Optimization: minimize a rewritten query

$Q_u:$

	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅]	[V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅]	[V _R ⁰ , ?r ₀ → ?p, ?r ₁ → ?r ₅]	[V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]
	[V _{FoF} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅]	[V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅]	[V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅]	[V _{FoF} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]
		[V _R ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]		[V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅]
		[V _{RoR} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]		[V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]

$V_F:$

CONSTRUCT{

- ① ?f₀ vfriend ?f₁,
- ② ?f₁ vives ?l₁}

WHERE{

?f₀ name Eric,
?f₀ friend ?f₁,
?f₁ lives ?l₁}

$V_{FoF}:$

CONSTRUCT{

- ① ?f₂ vfriend ?f₄,
- ② ?f₄ vives ?l₄}

WHERE{

?f₂ name Eric,
?f₂ friend ?f₃,
?f₃ friend ?f₄,
?f₄ lives ?l₄}

$V_R:$

CONSTRUCT{

- ① ?r₀ vrelated ?r₁,
- ② ?r₁ vives ?l₃}

WHERE{

?r₀ name Eric,
?r₀ related ?r₁,
?r₁ lives ?l₃}

$V_{RoR}:$

CONSTRUCT{

- ① ?r₂ vrelated ?r₄,
- ② ?r₄ vives ?l₆}

WHERE{

?r₂ name Eric,
?r₂ related ?r₃,
?r₃ related ?r₄,
?r₄ lives ?l₆}

Optimization: minimize a rewritten query

$Q_u:$

	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅]	[V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅]	[V _R ⁰ , ?r ₀ → ?p, ?r ₁ → ?r ₅]	[V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]
	[V _{FoF} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅]	[V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅]	[V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅]	[V _{FoF} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]
		[V _R ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]		[V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅]
		[V _{RoR} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]		[V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]

$V_F:$ CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ , ② ?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	$V_{FoF}:$ CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ , ② ?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	$V_R:$ CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ , ② ?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ }	$V_{RoR}:$ CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ , ② ?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }

Optimization: minimize a rewritten query

Rewriting q:

SELECT ?f₅ WHERE { ?p name Eric, ?p friend ?f₅, ?f₅ lives ?l₁, ?f'₀ name Eric, ?f'₀ friend ?f₅, ?f₅ lives ?l₅,
 ?p name Eric, ?p related ?r₅, ?r₅ lives ?l'₃, ?r'₂ name Eric, ?r'₂ related ?r'₃, ?r'₃ related ?r₅, ?r₅ lives ?l'₅}

$Q_u:$	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅ }
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅]	[V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅]	[V _R ⁰ , ?r ₀ → ?p, ?r ₁ → ?r ₅]	[V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]
	[V _{FoF} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅]	[V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅]	[V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅]	[V _{FoF} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]
		[V _R ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]		[V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅]
		[V _{RoR} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]		[V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]

$V_F:$	$V_{FoF}:$	$V_R:$	$V_{RoR}:$
CONSTRUCT{ $\textcircled{1}$?f ₀ vfriend ?f ₁ , $\textcircled{2}$?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	CONSTRUCT{ $\textcircled{1}$?f ₂ vfriend ?f ₄ , $\textcircled{2}$?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	CONSTRUCT{ $\textcircled{1}$?r ₀ vrelated ?r ₁ , $\textcircled{2}$?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ }	CONSTRUCT{ $\textcircled{1}$?r ₂ vrelated ?r ₄ , $\textcircled{2}$?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }

Optimization: minimize a rewritten query

Rewriting q:

SELECT ?f₅ WHERE { ?p name Eric, ?p friend ?f₅, ?f₅ lives ?l₁, ?f'₀ name Eric, ?f'₀ friend ?f₅, ?f₅ lives ?l₅,
?p name Eric, ?p related ?r₅, ?r₅ lives ?l'₃, ?r'₂ name Eric, ?r'₂ related ?r'₃, ?r'₃ related ?r₅, ?r₅ lives ?l₅}

Q _u :	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅]	[V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅]	[V _R ⁰ , ?r ₀ → ?p, ?r ₁ → ?r ₅]	[V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]
	[V _{FoF} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅]	[V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅]	[V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅]	[V _{FoF} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]
		[V _R ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]		[V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅]
		[V _{RoR} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]		[V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]

V _F :	V _{FoF} :	V _R :	V _{RoR} :
CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ , ② ?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ , ② ?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ , ② ?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ }	CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ , ② ?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }

- From the same view.

Optimization: minimize a rewritten query

Rewriting q:

SELECT ?f₅ WHERE { ?p name Eric, ?p friend ?f₅, ?f₅ lives ?l₁, ?f'₀ name Eric, ?f'₀ friend ?f₅, ?f₅ lives ?l₅,
?p name Eric, ?p related ?r₅, ?r₅ lives ?l'₃, ?r'₂ name Eric, ?r'₂ related ?r'₃, ?r'₃ related ?r₅, ?r₅ lives ?l'₅}

Q _u :	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅]	[V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅]	[V _R ⁰ , ?r ₀ → ?p, ?r ₁ → ?r ₅]	[V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]
	[V _{FoF} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅]	[V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅]	[V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅]	[V _{FoF} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]
		[V _R ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]		[V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅]
			[V _{RoR} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]	[V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]

V _F :	V _{FoF} :	V _R :	V _{RoR} :
CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ ② ?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ , ② ?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ , ② ?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ }	CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ , ② ?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }

- From the same view.
- Join in the same way in query and view.

Optimization: minimize a rewritten query

Rewriting q:

SELECT ?f₅ WHERE {
 ?p name Eric, ?p friend ?f₅, ?f₅ lives ?l₁, ?f'₀ name Eric, ?f'₀ friend ?f₅, ?f₅ lives ?l₅,
 ?p name Eric, ?p related ?r₅, ?r₅ lives ?l'₃, ?r'₂ name Eric, ?r'₂ related ?r'₃, ?r'₃ related ?r₅, ?r₅ lives ?l₅}

Q _u :	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅]	[V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅]	[V _R ⁰ , ?r ₀ → ?p, ?r ₁ → ?r ₅]	[V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]
	[V _{FoF} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅]	[V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅]	[V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅]	[V _{FoF} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]
		[V _R ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]		[V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅]
			[V _{RoR} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]	[V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]

V _F :	V _{FoF} :	V _R :	V _{RoR} :
CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ ② ?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ , ② ?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ , ② ?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ }	CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ , ② ?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }

- From the same view.
- Join in the same way in query and view.
- Can be merged.

Optimization: minimize a rewritten query

Rewriting q_{merge} :

SELECT ?f₅ WHERE {
 ?p name Eric, ?p friend ?f₅, ?f₅ lives ?l₅
 ?p name Eric, ?p related ?r₅, ?r₅ lives ?l'₃, ?r'₂ name Eric, ?r'₂ related ?r'₃, ?r'₃ related ?r₅, ?r₅ lives ?l₅}

Q _u :	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE {	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅]	[V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅]	[V _R ⁰ , ?r ₀ → ?p, ?r ₁ → ?r ₅]	[V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]
	[V _{FoF} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅]	[V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅]	[V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅]	[V _{FoF} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]
		[V _R ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]		[V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅]
			[V _{RoR} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]	[V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]

V _F :	V _{FoF} :	V _R :	V _{RoR} :
CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ ② ?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ , ② ?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ , ② ?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ }	CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ , ② ?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }

- From the same view.
- Join in the same way in query and view.
- Can be merged.

Optimization: minimize a rewritten query

- The proposed unification could potentially lead to a more efficient evaluation of the query.

Optimization: minimize a rewritten query

- The proposed unification could potentially lead to a more efficient evaluation of the query.
- We also show that such optimization will not affect the semantics of the query (sound and complete).

Optimization: minimize a rewritten query

- The proposed unification could potentially lead to a more efficient evaluation of the query.
- We also show that such optimization will not affect the semantics of the query (sound and complete).

Theorem

Query q_{merge} resulting from replacing the two copies of the same view in rewritten query q with one is equivalent to q .

Outline

1 Introduction

- Problem definition
- RDF data and SPARQL language
- A running example and solution outline

2 Rewriting on SPARQL views

- Rewriting Algorithm
- Remarks on SPARQL rewriting

3 Optimization on rewritings

- Minimize a rewritten query
- Pruning rewritings with empty results

4 Experiment

5 Conclusion

Optimization: pruning rewritings with empty results

- There are a subset of rewritings that join V_{FoF}^\otimes with V_R^\otimes .

$Q_u:$ SELECT ?f ₅ WHERE {	CandV ₁	CandV ₂	CandV ₃	CandV ₄
	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅] [V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅] [V _R ⁰ , r ₀ → ?p, ?r ₁ → ?r ₅] [V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]			
	[V _{FoF} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅] [V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅] [V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅] [V _{FoF} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]	[V _R ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]		[V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅]
			[V _{RoR} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]	[V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]

$V_F:$ CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ , ② ?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	$V_{FoF}:$ CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ , ② ?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	$V_R:$ CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ , ② ?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ }	$V_{RoR}:$ CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ , ② ?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }
---	---	---	--

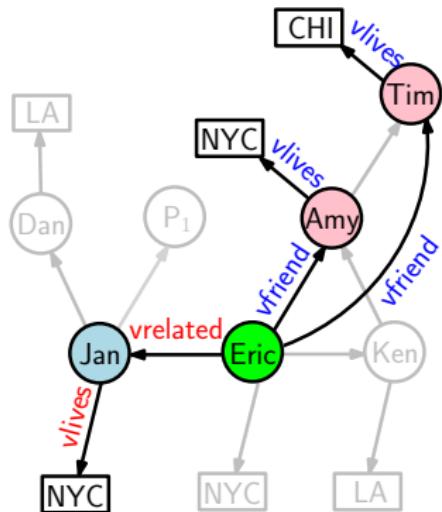
Optimization: pruning rewritings with empty results

- There are a subset of rewritings that join V_{FoF}^\otimes with V_R^\otimes .
- Looking for Eric's friend-of-friend who is also his relative. Who are they?

$Q_u:$ SELECT ?f ₅ WHERE {	CandV ₁	CandV ₂	CandV ₃	CandV ₄
	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅] [V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅] [V _R ⁰ , r ₀ → ?p, ?r ₁ → ?r ₅] [V _F ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]			
	[V _{FoF} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅] [V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅] [V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅] [V _{FoF} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]	[V _R ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]		[V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅]
			[V _{RoR} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]	[V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]

$V_F:$ CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ , ② ?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	$V_{FoF}:$ CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ , ② ?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	$V_R:$ CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ , ② ?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ }	$V_{RoR}:$ CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ , ② ?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }
---	---	---	--

Optimization: pruning rewritings with empty results



- It turns out that Eric's FoF and R are disjoint.

Optimization: pruning rewritings with empty results

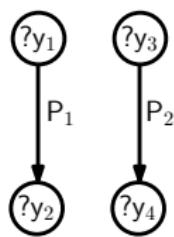
- There are a subset of rewritings that join V_{FoF}^\otimes with V_R^\otimes .
- Looking for Eric's friend-of-friend who is also his relative. Who are they?
- Prune all rewritings depending on the join between V_{FoF}^\otimes and V_R^\otimes .

$Q_u:$	CandV ₁	CandV ₂	CandV ₃	CandV ₄
$\text{SELECT } ?f_5 \text{ WHERE } \{$	$?p \text{ vfriend } ?f_5$	$?f_5 \text{ vives } ?l_5$	$?p \text{ vrelated } ?r_5$	$?r_5 \text{ vives } ?l_5$
	$[V_F^\otimes, ?f_0 \rightarrow ?p, ?f_1 \rightarrow ?f_5]$	$[V_F^\otimes, ?f_1 \rightarrow ?f_5, ?l_1 \rightarrow ?l_5]$	$[V_R^\otimes, r_0 \rightarrow ?p, ?r_1 \rightarrow ?r_5]$	$[V_F^\otimes, ?f_1 \rightarrow ?r_5, ?l_1 \rightarrow ?l_5]$
	$[V_{FoF}^\otimes, ?f_2 \rightarrow ?p, ?f_4 \rightarrow ?f_5]$	$[V_{FoF}^\otimes, ?f_4 \rightarrow ?f_5, ?l_4 \rightarrow ?l_5]$	$[V_{RoR}^\otimes, ?r_2 \rightarrow ?p, ?r_4 \rightarrow ?r_5]$	$[V_{FoF}^\otimes, ?f_4 \rightarrow ?r_5, ?l_4 \rightarrow ?l_5]$
		$[V_R^\otimes, ?r_1 \rightarrow ?f_5, ?l_3 \rightarrow ?l_5]$		$[V_R^\otimes, ?r_1 \rightarrow ?r_5, ?l_3 \rightarrow ?l_5]$
			$[V_{RoR}^\otimes, ?r_4 \rightarrow ?f_5, ?l_6 \rightarrow ?l_5]$	$[V_{RoR}^\otimes, ?r_4 \rightarrow ?r_5, ?l_6 \rightarrow ?l_5]$

$V_F:$	$V_{FoF}:$	$V_R:$	$V_{RoR}:$
$\text{CONSTRUCT}\{$	$\text{CONSTRUCT}\{$	$\text{CONSTRUCT}\{$	$\text{CONSTRUCT}\{$
$\textcircled{1} \ ?f_0 \text{ vfriend } ?f_1,$ $\textcircled{2} \ ?f_1 \text{ vives } ?l_1\}$	$\textcircled{1} \ ?f_2 \text{ vfriend } ?f_4,$ $\textcircled{2} \ ?f_4 \text{ vives } ?l_4\}$	$\textcircled{1} \ ?r_0 \text{ vrelated } ?r_1,$ $\textcircled{2} \ ?r_1 \text{ vives } ?l_3\}$	$\textcircled{1} \ ?r_2 \text{ vrelated } ?r_4,$ $\textcircled{2} \ ?r_4 \text{ vives } ?l_6\}$
$\text{WHERE}\{$ $?f_0 \text{ name Eric},$ $?f_0 \text{ friend } ?f_1,$ $?f_1 \text{ lives } ?l_1\}$	$\text{WHERE}\{$ $?f_2 \text{ name Eric},$ $?f_2 \text{ friend } ?f_3,$ $?f_3 \text{ friend } ?f_4,$ $?f_4 \text{ lives } ?l_4\}$	$\text{WHERE}\{$ $?r_0 \text{ name Eric},$ $?r_0 \text{ related } ?r_1,$ $?r_1 \text{ lives } ?l_3\}$	$\text{WHERE}\{$ $?r_2 \text{ name Eric},$ $?r_2 \text{ related } ?r_3,$ $?r_3 \text{ related } ?r_4,$ $?r_4 \text{ lives } ?l_6\}$

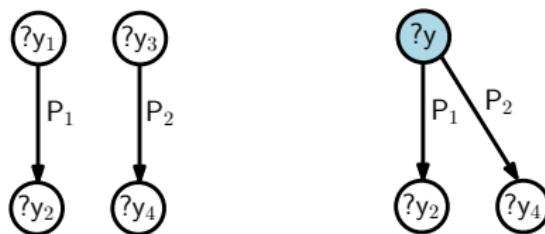
Optimization: pruning rewritings with empty results

- Consider a pair of triple patterns $(?y_1, p_1, ?y_2)$ and $(?y_3, p_2, ?y_4)$, suppose a join equals $?y_1$ and $?y_3$.



Optimization: pruning rewritings with empty results

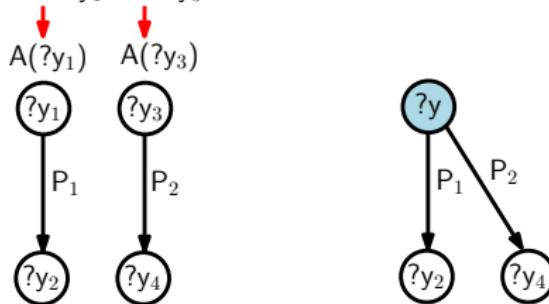
- Consider a pair of triple patterns $(?y_1, p_1, ?y_2)$ and $(?y_3, p_2, ?y_4)$, suppose a join equals $?y_1$ and $?y_3$.



Optimization: pruning rewritings with empty results

- Consider a pair of triple patterns $(?y_1, p_1, ?y_2)$ and $(?y_3, p_2, ?y_4)$, suppose a join equals $?y_1$ and $?y_3$.

Value sets of $?y_1$ and $?y_3$ in dataset.

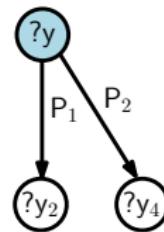
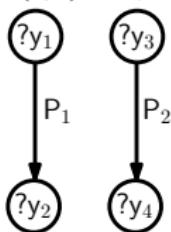


Optimization: pruning rewritings with empty results

- Consider a pair of triple patterns $(?y_1, p_1, ?y_2)$ and $(?y_3, p_2, ?y_4)$, suppose a join equals $?y_1$ and $?y_3$.

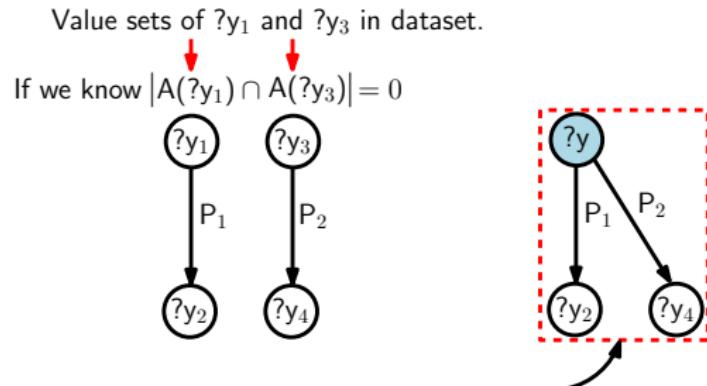
Value sets of $?y_1$ and $?y_3$ in dataset.

If we know $|A(?y_1) \cap A(?y_3)| = 0$



Optimization: pruning rewritings with empty results

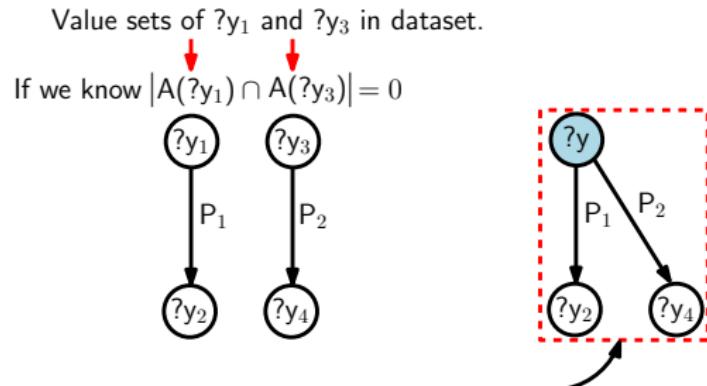
- Consider a pair of triple patterns $(?y_1, p_1, ?y_2)$ and $(?y_3, p_2, ?y_4)$, suppose a join equals $?y_1$ and $?y_3$.



- Any query with such join can be safely pruned.

Optimization: pruning rewritings with empty results

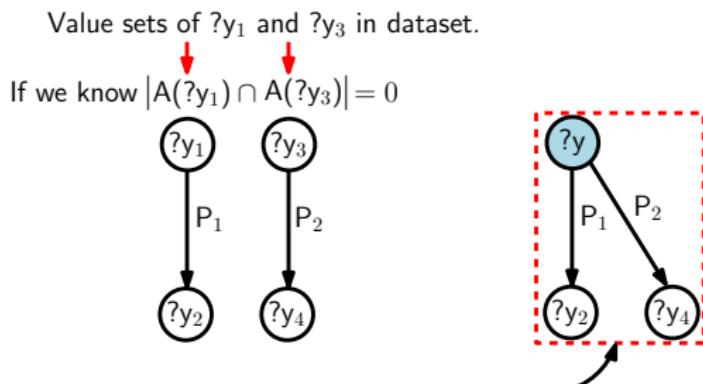
- Consider a pair of triple patterns $(?y_1, p_1, ?y_2)$ and $(?y_3, p_2, ?y_4)$, suppose a join equals $?y_1$ and $?y_3$.



- Any query with such join can be safely pruned.
- In theory, boolean intersection problem has been proven to require linear space.

Optimization: pruning rewritings with empty results

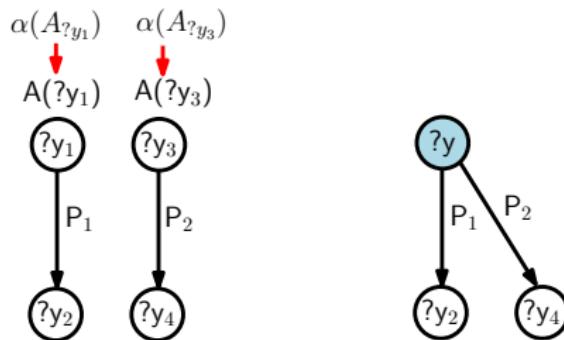
- Consider a pair of triple patterns $(?y_1, p_1, ?y_2)$ and $(?y_3, p_2, ?y_4)$, suppose a join equals $?y_1$ and $?y_3$.



- Any query with such join can be safely pruned.
- In theory, boolean intersection problem has been proven to require linear space.
- In practice, we can settle to a constant success probability and thus design a space-efficient heuristic.

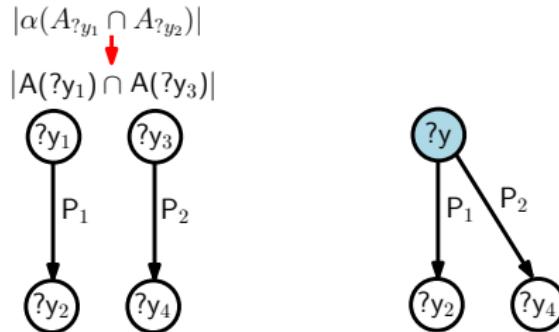
Optimization: pruning rewritings with empty results

- Instead of maintaining their exact value sets, keep two synapses for them, i.e., $\alpha(A_{?y_1})$ and $\alpha(A_{?y_3})$.



Optimization: pruning rewritings with empty results

- Instead of maintaining their exact value sets, keep two synapses for them, i.e., $\alpha(A_{?y_1})$ and $\alpha(A_{?y_3})$.

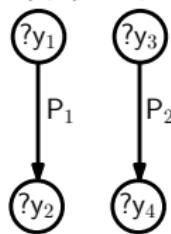


Optimization: pruning rewritings with empty results

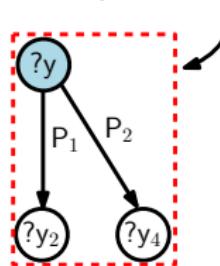
- Instead of maintaining their exact value sets, keep two synapses for them, i.e., $\alpha(A_{?y_1})$ and $\alpha(A_{?y_3})$.

$$|\alpha(A_{?y_1} \cap A_{?y_2})| > \tau$$

$$|A(?y_1) \cap A(?y_3)|$$

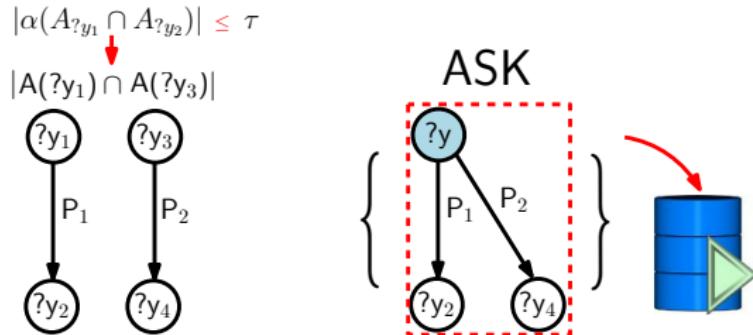


Continue rewriting with the pattern.



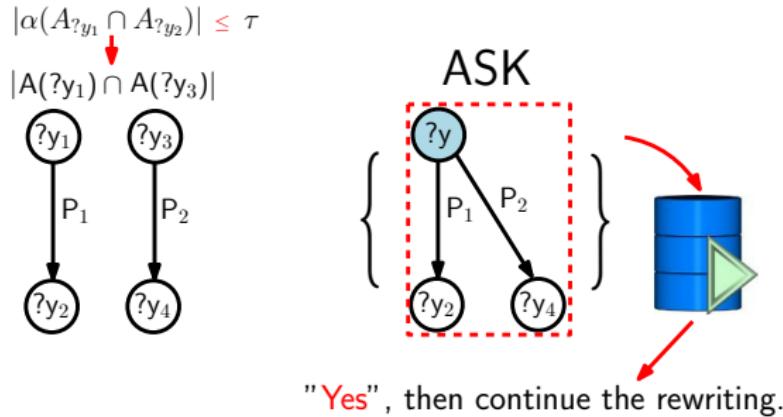
Optimization: pruning rewritings with empty results

- Instead of maintaining their exact value sets, keep two synopses for them, i.e., $\alpha(A_{?y_1})$ and $\alpha(A_{?y_3})$.



Optimization: pruning rewritings with empty results

- Instead of maintaining their exact value sets, keep two synopses for them, i.e., $\alpha(A_{?y_1})$ and $\alpha(A_{?y_3})$.

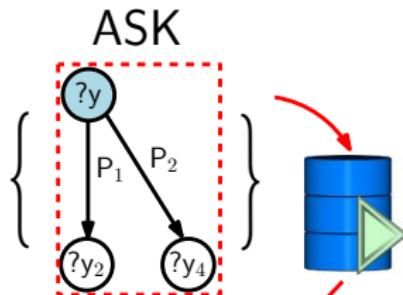
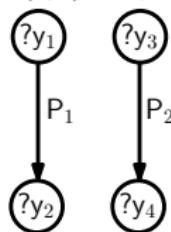


Optimization: pruning rewritings with empty results

- Instead of maintaining their exact value sets, keep two synopses for them, i.e., $\alpha(A_{?y_1})$ and $\alpha(A_{?y_3})$.

$$|\alpha(A_{?y_1} \cap A_{?y_2})| \leq \tau$$

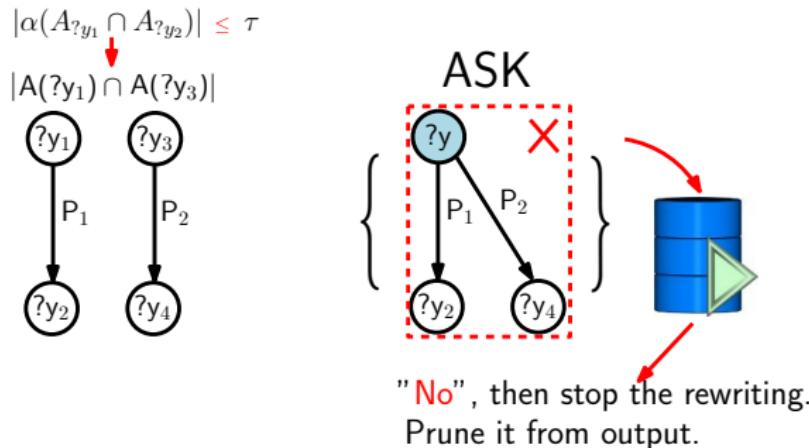
$$|A(?y_1) \cap A(?y_3)|$$



"No", then stop the rewriting.

Optimization: pruning rewritings with empty results

- Instead of maintaining their exact value sets, keep two synopses for them, i.e., $\alpha(A_{?y_1})$ and $\alpha(A_{?y_3})$.



*In the paper, we presented heuristics to avoid issuing excessive ASK queries in rewriting.

Optimization: pruning rewritings with empty results

- Instead of maintaining their exact value sets, keep two synopses for them, *i.e.*, $\alpha(A?_{y_1})$ and $\alpha(A?_{y_3})$.
- In practice, we use K Minimum Value (KMV) synopses to obtain unbiased estimations for the size of join.

Optimization: pruning rewritings with empty results

- Instead of maintaining their exact value sets, keep two synopses for them, *i.e.*, $\alpha(A?_{y_1})$ and $\alpha(A?_{y_3})$.
- In practice, we use K Minimum Value (KMV) synopses to obtain unbiased estimations for the size of join.
- In the paper, we also discuss how to maintain the synopses, *i.e.*, dealing with updates.

Outline

1 Introduction

- Problem definition
- RDF data and SPARQL language
- A running example and solution outline

2 Rewriting on SPARQL views

- Rewriting Algorithm
- Remarks on SPARQL rewriting

3 Optimization on rewritings

- Minimize a rewritten query
- Pruning rewritings with empty results

4 Experiment

5 Conclusion

Experiments

- Implementation: We implemented the rewriting algorithms and optimizations in C++.
- Views and Queries: We defined queries and views on LUBM benchmark (e.g. about courses, students, professors).
- Data: 10M triples from LUBM.
- Platform: 64-bit linux with 2GHz Intel Xeon CPU and 4GB Mem.
- Store: Evaluations were performed on **MySQL** (for relational exp) and **4store** (a generic RDF store).

Experiments: Notation

- SPARQL query rewriting without optimizations as **SQR**.
- SPARQL query rewriting with optimizations as **OSQR**.
- Translation to SQL and rewriting in relational DB as **SQL**.
- Refer to our paper for the full set of experiments.

Experiments: Setup

- We create 7 types of views for 14 persons respectively.

View About	1	2	3	4	5	6	7
Name							
Person 1							
Person 2							
Person 3							NULL
Person 4							NULL
Person 5						NULL	NULL
Person 6						NULL	NULL
Person 7					NULL	NULL	NULL
Person 8				NULL	NULL	NULL	NULL
Person 9			NULL	NULL	NULL	NULL	NULL
Person 10			NULL	NULL	NULL	NULL	NULL
Person 11		NULL	NULL	NULL	NULL	NULL	NULL
Person 12		NULL	NULL	NULL	NULL	NULL	NULL
Person 13	NULL						
Person 14	NULL						

Experiments: Setup

- We create 7 types of views for 14 persons respectively.
- Each person releases a portion of his/her 7 views for querying.

	1	2	3	4	5	6	7
View About	Name	Email	DegreeFrom	Phone	TeacherOf	Interest	WorkFor
Person 1							
Person 2							
Person 3							NULL
Person 4							NULL
Person 5						NULL	NULL
Person 6						NULL	NULL
Person 7					NULL	NULL	NULL
Person 8					NULL	NULL	NULL
Person 9				NULL	NULL	NULL	NULL
Person 10				NULL	NULL	NULL	NULL
Person 11			NULL	NULL	NULL	NULL	NULL
Person 12			NULL	NULL	NULL	NULL	NULL
Person 13	NULL	NULL	NULL	NULL	NULL	NULL	NULL
Person 14	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Experiments: Setup

- We create 7 types of views for 14 persons respectively.
- Each person releases a portion of his/her 7 views for querying.

Person 1 publish all his/her views.

View About	1	2	3	4	5	6	7
Name							
Person 1							
Person 2							
Person 3							NULL
Person 4							NULL
Person 5						NULL	NULL
Person 6						NULL	NULL
Person 7					NULL	NULL	NULL
Person 8					NULL	NULL	NULL
Person 9				NULL	NULL	NULL	NULL
Person 10				NULL	NULL	NULL	NULL
Person 11			NULL	NULL	NULL	NULL	NULL
Person 12			NULL	NULL	NULL	NULL	NULL
Person 13	NULL						
Person 14	NULL						

Experiments: Setup

- We create 7 types of views for 14 persons respectively.
- Each person releases a portion of his/her 7 views for querying.

View About	1	2	3	4	5	6	7
Name							
Person 1							
Person 2							
Person 3							NULL
Person 4							NULL
Person 5						NULL	NULL
Person 6						NULL	NULL
Person 7					NULL	NULL	NULL
Person 8					NULL	NULL	NULL
Person 9				NULL	NULL	NULL	NULL
Person 10				NULL	NULL	NULL	NULL
Person 11			NULL	NULL	NULL	NULL	NULL
Person 12			NULL	NULL	NULL	NULL	NULL
Person 13	NULL						
Person 14	NULL						

Person 14 only published the view about name.

Experiments: Setup

- We create 7 types of views for 14 persons respectively.
- Each person releases a portion of his/her 7 views for querying.

	1	2	3	4	5	6	7
View About	Name	Email	DegreeFrom	Phone	TeacherOf Interest	WorkFor	
Person 1							
Person 2							
Person 3							NULL
Person 4							NULL
Person 5						NULL	NULL
Person 6						NULL	NULL
Person 7					NULL	NULL	NULL
Person 8					NULL	NULL	NULL
Person 9				NULL	NULL	NULL	NULL
Person 10				NULL	NULL	NULL	NULL
Person 11			NULL	NULL	NULL	NULL	NULL
Person 12			NULL	NULL	NULL	NULL	NULL
Person 13	NULL	NULL	NULL	NULL	NULL	NULL	NULL
Person 14	NULL	NULL	NULL	NULL	NULL	NULL	NULL

$Q_u:$ `SELECT * {
 ①?x name ?n, ②?x email ?e, ③?x DegreeFrom ?d,`

- A query incrementally asks for more properties.

Experiments: Setup

- We create 7 types of views for 14 persons respectively.
- Each person releases a portion of his/her 7 views for querying.

	1	2	3	4	5	6	7
View About	Name	Email	DegreeFrom	Phone	TeacherOf Interest	WorkFor	
Person 1							
Person 2							
Person 3							NULL
Person 4							NULL
Person 5						NULL	NULL
Person 6						NULL	NULL
Person 7					NULL	NULL	NULL
Person 8					NULL	NULL	NULL
Person 9				NULL	NULL	NULL	NULL
Person 10				NULL	NULL	NULL	NULL
Person 11			NULL	NULL	NULL	NULL	NULL
Person 12			NULL	NULL	NULL	NULL	NULL
Person 13	NULL	NULL	NULL	NULL	NULL	NULL	NULL
Person 14	NULL	NULL	NULL	NULL	NULL	NULL	NULL

$Q_u:$ SELECT * {

①?x name ?n, ②?x email ?e, ③?x DegreeFrom ?d, ④?x phone ?p,

- }
- A query incrementally asks for more properties.

Experiments: Setup

- We create 7 types of views for 14 persons respectively.
- Each person releases a portion of his/her 7 views for querying.

View About	1	2	3	4	5	6	7
Person 1							
Person 2							
Person 3							NULL
Person 4							NULL
Person 5						NULL	NULL
Person 6						NULL	NULL
Person 7					NULL	NULL	NULL
Person 8				NULL	NULL	NULL	NULL
Person 9			NULL	NULL	NULL	NULL	NULL
Person 10			NULL	NULL	NULL	NULL	NULL
Person 11		NULL	NULL	NULL	NULL	NULL	NULL
Person 12		NULL	NULL	NULL	NULL	NULL	NULL
Person 13	NULL						
Person 14	NULL						

$Q_u:$ SELECT * {
 ①?x name ?n, ②?x email ?e, ③?x DegreeFrom ?d, ④?x phone ?p,
 ⑤?x teacherOf ?t,
}

- A query incrementally asks for more properties.

Experiments: Setup

- We create 7 types of views for 14 persons respectively.
- Each person releases a portion of his/her 7 views for querying.

	1	2	3	4	5	6	7
View About	Name	Email	DegreeFrom	Phone	TeacherOf Interest	WorkFor	
Person 1							
Person 2							
Person 3							NULL
Person 4							NULL
Person 5						NULL	NULL
Person 6						NULL	NULL
Person 7					NULL	NULL	NULL
Person 8					NULL	NULL	NULL
Person 9				NULL	NULL	NULL	NULL
Person 10				NULL	NULL	NULL	NULL
Person 11			NULL	NULL	NULL	NULL	NULL
Person 12			NULL	NULL	NULL	NULL	NULL
Person 13	NULL	NULL	NULL	NULL	NULL	NULL	NULL
Person 14	NULL	NULL	NULL	NULL	NULL	NULL	NULL

$Q_u:$ SELECT * {
 ①?x name ?n, ②?x email ?e, ③?x DegreeFrom ?d, ④?x phone ?p,
 ⑤?x teacherOf ?t, ⑥?x interest ?i,
}

- A query incrementally asks for more properties.

Experiments: Setup

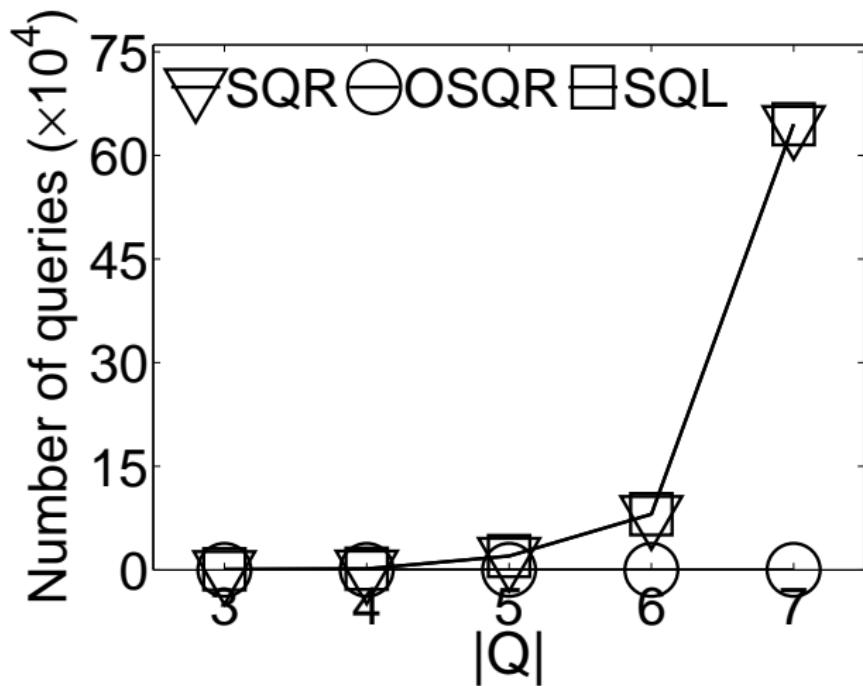
- We create 7 types of views for 14 persons respectively.
- Each person releases a portion of his/her 7 views for querying.

View About	1	2	3	4	5	6	7
Person 1							
Person 2							
Person 3							NULL
Person 4							NULL
Person 5						NULL	NULL
Person 6						NULL	NULL
Person 7					NULL	NULL	NULL
Person 8					NULL	NULL	NULL
Person 9				NULL	NULL	NULL	NULL
Person 10				NULL	NULL	NULL	NULL
Person 11			NULL	NULL	NULL	NULL	NULL
Person 12			NULL	NULL	NULL	NULL	NULL
Person 13	NULL						
Person 14	NULL						

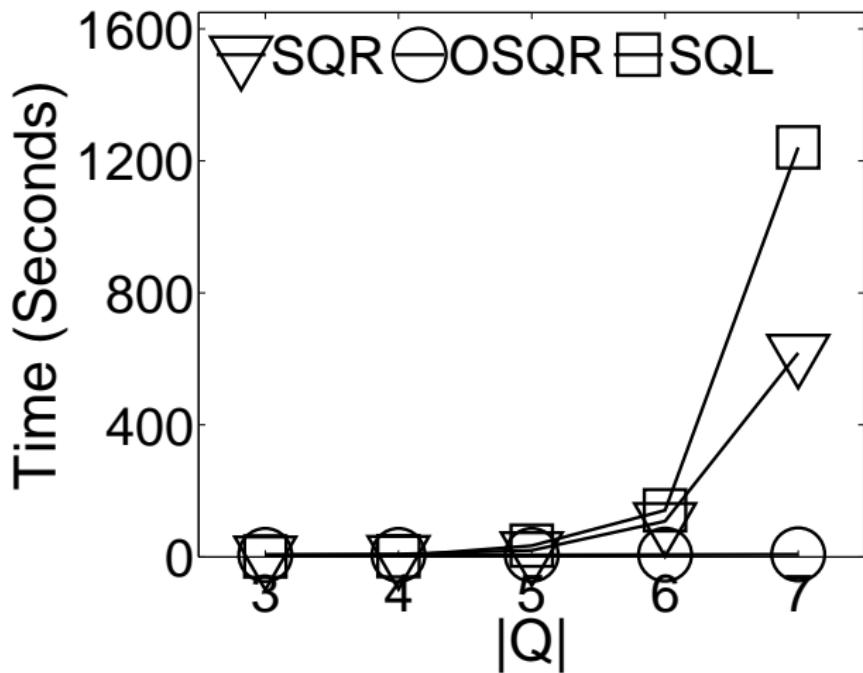
$Q_u:$ SELECT * {
 ①?x name ?n, ②?x email ?e, ③?x DegreeFrom ?d, ④?x phone ?p,
 ⑤?x teacherOf ?t, ⑥?x interest ?i, ⑦?x worksFor ?w.
}

- A query incrementally asks for more properties.

Experiments: Results



Experiments: Results



Experiments: Setup

	Email Course	Phone Course	Degree Course	Email Course	Email Course
Template	T1	T2	T3	T4	T5
Univ 1		NULL	NULL	NULL	NULL
Univ 2	NULL		NULL	NULL	NULL
Univ 3	NULL	NULL		NULL	
Univ 4	NULL	NULL	NULL	NULL	

Experiments: Setup

	Email Course	Phone Course	Degree Course	Email Course	Email Course
Template	T1	T2	T3	T4	T5
Univ 1		NULL	NULL	NULL	NULL
Univ 2	NULL		NULL	NULL	NULL
Univ 3	NULL	NULL		NULL	
Univ 4	NULL	NULL	NULL	NULL	
Univ 5		NULL	NULL	NULL	NULL

Experiments: Setup

	Email Course	Phone Course	Degree Course	Email Course	Email Course
Template	T1	T2	T3	T4	T5
Univ 1	NULL	NULL	NULL	NULL	NULL
Univ 2	NULL	NULL	NULL	NULL	NULL
Univ 3	NULL	NULL	NULL	NULL	NULL
Univ 4	NULL	NULL	NULL	NULL	NULL
Univ 5	NULL	NULL	NULL	NULL	NULL
Univ 6	NULL	NULL	NULL	NULL	NULL

Experiments: Setup

	Email Course	Phone Course	Degree Course	Email Course	Email Course
Template	T1	T2	T3	T4	T5
Univ 1		NULL	NULL	NULL	NULL
Univ 2	NULL		NULL	NULL	NULL
Univ 3	NULL	NULL		NULL	
Univ 4	NULL	NULL	NULL	NULL	
Univ 5		NULL	NULL	NULL	NULL
Univ 6	NULL		NULL	NULL	NULL
Univ 7	NULL	NULL		NULL	

Experiments: Setup

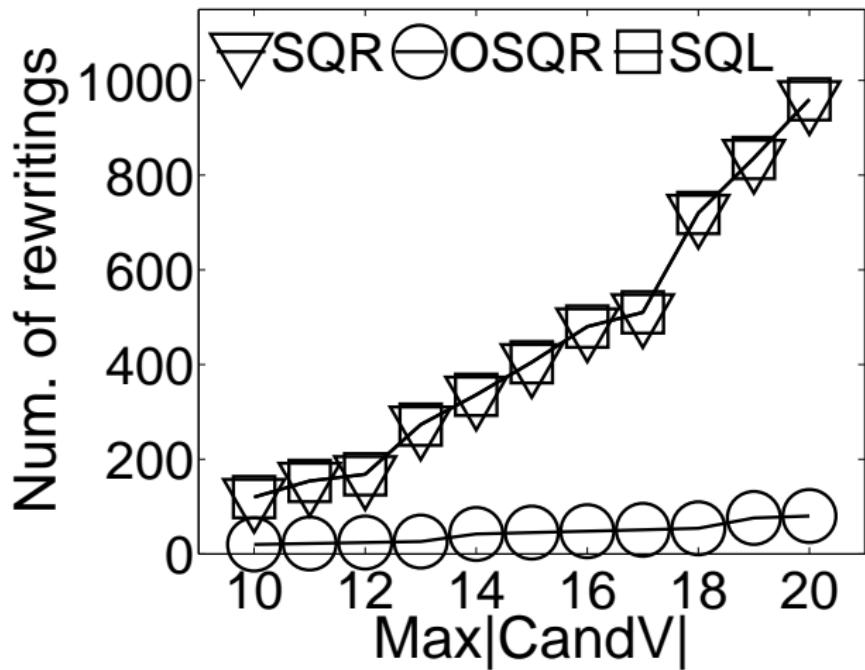
	Email Course	Phone Course	Degree Course	Email Course	Email Course
Template	T1	T2	T3	T4	T5
Univ 1		NULL	NULL	NULL	NULL
Univ 2	NULL		NULL	NULL	NULL
Univ 3	NULL	NULL			NULL
Univ 4	NULL	NULL	NULL	NULL	
Univ 5		NULL	NULL	NULL	NULL
Univ 6	NULL		NULL	NULL	NULL
Univ 7	NULL	NULL			NULL
Univ 8	NULL	NULL	NULL	NULL	
Univ 9		NULL	NULL	NULL	NULL
Univ 10	NULL		NULL	NULL	NULL
Univ 11	NULL	NULL			NULL
Univ 12	NULL	NULL	NULL	NULL	
Univ 13		NULL	NULL	NULL	NULL
Univ 14	NULL		NULL	NULL	NULL
Univ 15	NULL	NULL			NULL
Univ 16	NULL	NULL	NULL	NULL	
Univ 17		NULL	NULL	NULL	NULL
Univ 18	NULL		NULL	NULL	NULL
Univ 19	NULL	NULL			NULL
Univ 20	NULL	NULL	NULL	NULL	

Experiments: Setup

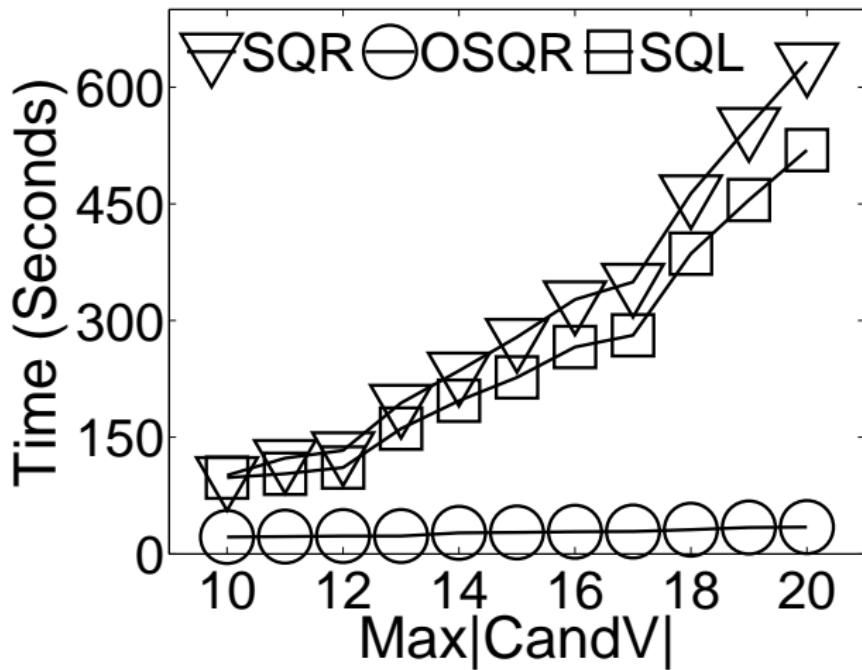
	Email Course	Phone Course	Degree Course	Email Course	Email Course
Template	T1	T2	T3	T4	T5
Univ 1		NULL	NULL	NULL	NULL
Univ 2	NULL		NULL	NULL	NULL
Univ 3	NULL	NULL			NULL
Univ 4	NULL	NULL	NULL	NULL	
Univ 5		NULL	NULL	NULL	NULL
Univ 6	NULL		NULL	NULL	NULL
Univ 7	NULL	NULL			NULL
Univ 8	NULL	NULL	NULL	NULL	
Univ 9		NULL	NULL	NULL	NULL
Univ 10	NULL		NULL	NULL	NULL
Univ 11	NULL	NULL			NULL
Univ 12	NULL	NULL	NULL	NULL	
Univ 13		NULL	NULL	NULL	NULL
Univ 14	NULL		NULL	NULL	NULL
Univ 15	NULL	NULL			NULL
Univ 16	NULL	NULL	NULL	NULL	
Univ 17		NULL	NULL	NULL	NULL
Univ 18	NULL		NULL	NULL	NULL
Univ 19	NULL	NULL			NULL
Univ 20	NULL	NULL	NULL	NULL	

Q_u: SELECT * {
 ①?x course ?n, ②?x email ?e,
 ③?x degree ?d,
}

Experiments: Results



Experiments: Results



Outline

1 Introduction

- Problem definition
- RDF data and SPARQL language
- A running example and solution outline

2 Rewriting on SPARQL views

- Rewriting Algorithm
- Remarks on SPARQL rewriting

3 Optimization on rewritings

- Minimize a rewritten query
- Pruning rewritings with empty results

4 Experiment

5 Conclusion

Conclusion

- We studied rewriting queries over views in the context of SPARQL and RDF.
- We proposed the first *sound* and *complete* rewriting algorithm.
- Novel optimizations to simplify individual rewritings and prune rewritings with empty results.
- Our solution is independent of RDF stores and hence portable.
- Extensive experiments demonstrate that our method is efficient and scalable.

The End

Thank You

Q and A

Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.

Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.

$Q_u:$ SELECT ?f ₅ WHERE {	CandV ₁	CandV ₂	CandV ₃	CandV ₄
	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _F ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅] [V _F ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅] [V _{F,R,F} ⁰ , ?r ₀ → ?p, ?r ₁ → ?r ₅] [V _{F,R,F} ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]			
	[V _{F,o,F} ⁰ , ?f ₂ → ?p, ?f ₄ → ?f ₅] [V _{F,o,F} ⁰ , ?f ₄ → ?f ₅ , ?l ₃ → ?l ₅] [V _{R,o,R} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅] [V _{F,o,F} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]	[V _F ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]	[V _R ⁰ , ?r ₁ → ?r ₅ , ?l ₃ → ?l ₅]	
	[V _{R,o,R} ⁰ , ?r ₄ → ?f ₅ , ?l ₆ → ?l ₅]		[V _{R,o,R} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]	
$V_F:$ CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ . ② ?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ }	$V_{F,o,F}:$ CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ . ② ?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ }	$V_R:$ CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ . ② ?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ }	$V_{R,o,R}:$ CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ . ② ?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ }	

Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.

Queried Variable	CandV1	CandV2	CandV3	CandV4
SELECT ?f ₅ WHERE { ?p vfriend ?f ₅ ?f ₅ vlivs ?l ₅ ?p vrelated ?r ₅ ?f ₅ vlivs ?l ₅ }	{?p vfriend ?f ₅ ?f ₅ vlivs ?l ₅ ?p vrelated ?r ₅ ?f ₅ vlivs ?l ₅ }	{?p vfriend ?f ₅ ?f ₅ vlivs ?l ₅ ?p vrelated ?r ₅ ?f ₅ vlivs ?l ₅ }	{?p vfriend ?f ₅ ?f ₅ vlivs ?l ₅ ?p vrelated ?r ₅ ?f ₅ vlivs ?l ₅ }	{?p vfriend ?f ₅ ?f ₅ vlivs ?l ₅ ?p vrelated ?r ₅ ?f ₅ vlivs ?l ₅ }

$V_F:$	$V_{FOF}:$	$V_R:$	$V_{RoR}:$
CONSTRUCT{	CONSTRUCT{	CONSTRUCT{	CONSTRUCT{
① $?_6 \text{ vfriend } ?_1,$	① $?_7 \text{ vfriend } ?_4,$	① $?_0 \text{ related } ?_1,$	① $?_2 \text{ vrelated } ?_4,$
② $?_1 \text{ vives } ?_1,$	② $?_3 \text{ vives } ?_4,$	② $?_1 \text{ vives } ?_3,$	② $?_3 \text{ vives } ?_6,$
WHERE{	WHERE{	WHERE{	WHERE{
?6 name Eric, ?6 friend ?1, ?1 lives ?1}	?7 name Eric, ?7 friend ?3, ?3 friend ?4, ?4 lives ?4}	?0 name Eric, ?0 related ?1, ?1 lives ?3}	?2 name Eric, ?2 related ?3, ?3 related ?4, ?4 lives ?6}

Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.

Start with the smallest $|CandV|$.

Q_u:

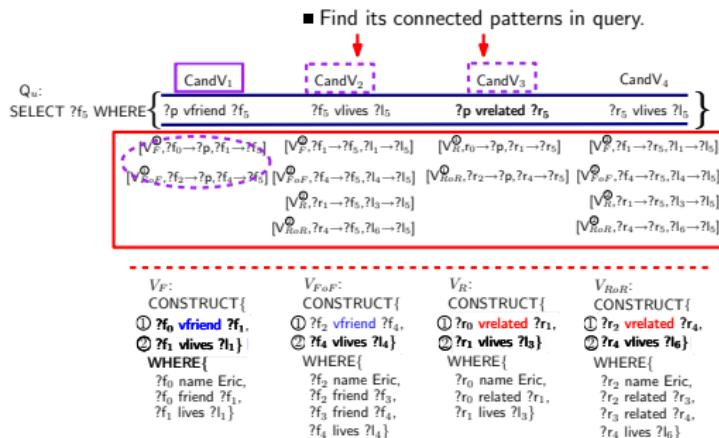
SELECT ?f₅ WHERE { ?p vfriend ?f₅ }
 { ?f₅ vives ?l₅ }
 { ?p vrelated ?r₅ }
 { ?r₅ vives ?l₅ }

↓

CandV ₁	CandV ₂	CandV ₃	CandV ₄
$\nabla_{P,F}^0$: CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ . ② ?f ₁ vives ?l ₁ } WHERE{ ?f ₀ name Eric. ?f ₀ friend ?f ₁ . ?f ₁ lives ?l ₁ }	$\nabla_{P,F}^0$: CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ . ② ?f ₄ vives ?l ₄ } WHERE{ ?f ₂ name Eric. ?f ₂ friend ?f ₃ . ?f ₃ friend ?f ₄ . ?f ₄ lives ?l ₄ }	$\nabla_{R,F}^0$: CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ . ② ?r ₁ vives ?l ₃ } WHERE{ ?r ₀ name Eric. ?r ₀ related ?r ₁ . ?r ₁ lives ?l ₃ }	$\nabla_{R,R}^0$: CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ . ② ?r ₄ vives ?l ₆ } WHERE{ ?r ₂ name Eric. ?r ₂ related ?r ₃ . ?r ₃ related ?r ₄ . ?r ₄ lives ?l ₆ }
$\nabla_{P,F,R}^0$: CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ . ② ?f ₁ vives ?l ₁ . ③ ?r ₀ vrelated ?r ₁ . ④ ?r ₁ vives ?l ₃ } WHERE{ ?f ₀ name Eric. ?f ₀ friend ?f ₁ . ?f ₁ lives ?l ₁ . ?r ₀ name Eric. ?r ₀ related ?r ₁ . ?r ₁ lives ?l ₃ }	$\nabla_{P,F,R}^0$: CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ . ② ?f ₄ vives ?l ₄ . ③ ?r ₂ vrelated ?r ₄ . ④ ?r ₄ vives ?l ₆ } WHERE{ ?f ₂ name Eric. ?f ₂ friend ?f ₃ . ?f ₃ friend ?f ₄ . ?f ₄ lives ?l ₄ . ?r ₂ name Eric. ?r ₂ related ?r ₃ . ?r ₃ related ?r ₄ . ?r ₄ lives ?l ₆ }	$\nabla_{R,R,F}^0$: CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ . ② ?r ₁ vives ?l ₃ . ③ ?r ₂ vrelated ?r ₄ . ④ ?r ₄ vives ?l ₆ } WHERE{ ?r ₀ name Eric. ?r ₀ related ?r ₁ . ?r ₁ lives ?l ₃ . ?r ₂ name Eric. ?r ₂ related ?r ₃ . ?r ₃ related ?r ₄ . ?r ₄ lives ?l ₆ }	$\nabla_{P,F,R,R}^0$: CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ . ② ?f ₁ vives ?l ₁ . ③ ?r ₀ vrelated ?r ₁ . ④ ?r ₁ vives ?l ₃ . ⑤ ?r ₂ vrelated ?r ₄ . ⑥ ?r ₄ vives ?l ₆ } WHERE{ ?f ₀ name Eric. ?f ₀ friend ?f ₁ . ?f ₁ lives ?l ₁ . ?r ₀ name Eric. ?r ₀ related ?r ₁ . ?r ₁ lives ?l ₃ . ?r ₂ name Eric. ?r ₂ related ?r ₃ . ?r ₃ related ?r ₄ . ?r ₄ lives ?l ₆ }

Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.



Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.

Q _u :	CandV ₁	CandV ₂	CandV ₃	CandV ₄
SELECT ?f ₅ WHERE { ?p vfriend ?f ₅ } { ?f ₅ vives ?l ₅ } { ?p vrelated ?r ₅ } { ?r ₅ vives ?l ₅ }	{ [V _{P,f0} ?f ₀ →?p,?f ₁ →?r ₁] } { [V _{P,f1} ?f ₁ →?p,?f ₂ →?r ₂] } { [V _{P,f2} ?f ₂ →?p,?f ₃ →?r ₃] } { [V _{P,f3} ?f ₃ →?p,?f ₄ →?r ₄] } { [V _{P,f4} ?f ₄ →?p,?f ₅ ,?l ₄ →?l ₅] } { [V _{R,f0R} ?r ₀ →?r ₁ ,?l ₀ →?l ₁] } { [V _{R,f1R} ?r ₁ →?r ₂ ,?l ₁ →?l ₂] } { [V _{R,f2R} ?r ₂ →?r ₃ ,?l ₂ →?l ₃] } { [V _{R,f3R} ?r ₃ →?r ₄ ,?l ₃ →?l ₄] } { [V _{R,f4R} ?r ₄ →?r ₅ ,?l ₄ →?l ₅] }	{ [V _{P,f0} ?f ₀ →?p,?f ₁ →?r ₁] } { [V _{P,f1} ?f ₁ →?p,?f ₂ →?r ₂] } { [V _{P,f2} ?f ₂ →?p,?f ₃ →?r ₃] } { [V _{P,f3} ?f ₃ →?p,?f ₄ →?r ₄] } { [V _{P,f4} ?f ₄ →?p,?f ₅ ,?l ₄ →?l ₅] } { [V _{R,f0R} ?r ₀ →?r ₁ ,?l ₀ →?l ₁] } { [V _{R,f1R} ?r ₁ →?r ₂ ,?l ₁ →?l ₂] } { [V _{R,f2R} ?r ₂ →?r ₃ ,?l ₂ →?l ₃] } { [V _{R,f3R} ?r ₃ →?r ₄ ,?l ₃ →?l ₄] } { [V _{R,f4R} ?r ₄ →?r ₅ ,?l ₄ →?l ₅] }	{ [V _{P,f0} ?f ₀ →?p,?f ₁ →?r ₁] } { [V _{P,f1} ?f ₁ →?p,?f ₂ →?r ₂] } { [V _{P,f2} ?f ₂ →?p,?f ₃ →?r ₃] } { [V _{P,f3} ?f ₃ →?p,?f ₄ →?r ₄] } { [V _{P,f4} ?f ₄ →?p,?f ₅ ,?l ₄ →?l ₅] } { [V _{R,f0R} ?r ₀ →?r ₁ ,?l ₀ →?l ₁] } { [V _{R,f1R} ?r ₁ →?r ₂ ,?l ₁ →?l ₂] } { [V _{R,f2R} ?r ₂ →?r ₃ ,?l ₂ →?l ₃] } { [V _{R,f3R} ?r ₃ →?r ₄ ,?l ₃ →?l ₄] } { [V _{R,f4R} ?r ₄ →?r ₅ ,?l ₄ →?l ₅] }	{ [V _{P,f0} ?f ₀ →?p,?f ₁ →?r ₁] } { [V _{P,f1} ?f ₁ →?p,?f ₂ →?r ₂] } { [V _{P,f2} ?f ₂ →?p,?f ₃ →?r ₃] } { [V _{P,f3} ?f ₃ →?p,?f ₄ →?r ₄] } { [V _{P,f4} ?f ₄ →?p,?f ₅ ,?l ₄ →?l ₅] } { [V _{R,f0R} ?r ₀ →?r ₁ ,?l ₀ →?l ₁] } { [V _{R,f1R} ?r ₁ →?r ₂ ,?l ₁ →?l ₂] } { [V _{R,f2R} ?r ₂ →?r ₃ ,?l ₂ →?l ₃] } { [V _{R,f3R} ?r ₃ →?r ₄ ,?l ₃ →?l ₄] } { [V _{R,f4R} ?r ₄ →?r ₅ ,?l ₄ →?l ₅] }
V _P : CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ . ② ?f ₁ vives ?l ₁ .} WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ .}	V _{P,f0P} : CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ . ② ?f ₄ vives ?l ₄ .} WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ .}	V _R : CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ . ② ?r ₁ vives ?l ₃ .} WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ .}	V _{R,R} : CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ . ② ?r ₄ vives ?l ₆ .} WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ .}	

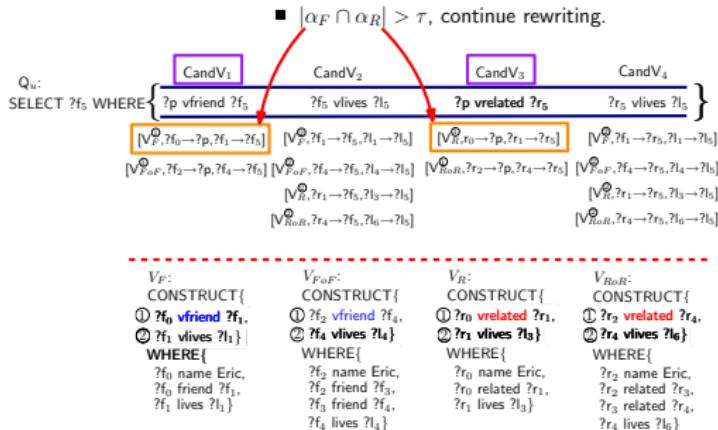
Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.

$Q_u:$ SELECT ?f ₅ WHERE {	CandV ₁	CandV ₂	CandV ₃	CandV ₄
	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _{FoF} ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅] [V _{FoF} ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅] [V _{FoF} ⁰ , ?f ₀ → ?p, ?r ₁ → ?r ₅] [V _{FoF} ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]	[V _{FoF} ⁰ , ?f ₀ → ?p, ?f ₄ → ?f ₅] [V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅] [V _{FoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅] [V _{FoR} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]	[V _{FoF} ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅ , ?l ₀ → ?l ₅] [V _{FoF} ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅] [V _{FoR} ⁰ , ?r ₂ → ?p, ?r ₁ → ?r ₅] [V _{FoR} ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]	[V _{FoF} ⁰ , ?f ₀ → ?p, ?f ₄ → ?f ₅ , ?l ₀ → ?l ₅] [V _{FoF} ⁰ , ?f ₄ → ?f ₅ , ?l ₄ → ?l ₅] [V _{FoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅] [V _{FoR} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]
	V _P : CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ . ② ?f ₁ vives ?l ₁ .} WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ .}	V _{FoF} : CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ . ② ?f ₄ vives ?l ₄ .} WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ .}	V _R : CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ . ② ?r ₁ vives ?l ₃ .} WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ .}	V _{FoR} : CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ . ② ?r ₄ vives ?l ₆ .} WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ .}

Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.



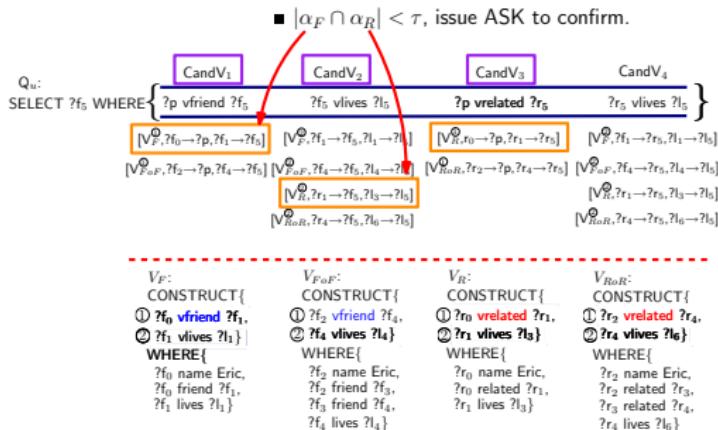
Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.

$Q_u:$ SELECT ?f ₅ WHERE {	CandV ₁	CandV ₂	CandV ₃	CandV ₄
	?p vfriend ?f ₅	?f ₅ vives ?l ₅	?p vrelated ?r ₅	?r ₅ vives ?l ₅
	[V _P ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅] [V _P ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅] [V _P ⁰ , ?r ₀ → ?p, ?r ₁ → ?r ₅] [V _P ⁰ , ?f ₁ → ?r ₅ , ?l ₁ → ?l ₅]	[V _P ⁰ , ?f ₀ → ?p, ?f ₄ → ?f ₅] [V _P ⁰ , ?f ₄ → ?f ₅ , ?l ₃ → ?l ₅] [V _{RoR} ⁰ , ?r ₂ → ?p, ?r ₄ → ?r ₅] [V _{RoR} ⁰ , ?f ₄ → ?r ₅ , ?l ₄ → ?l ₅]	[V _P ⁰ , ?f ₀ → ?p, ?f ₄ → ?f ₅] [V _P ⁰ , ?f ₄ → ?f ₅ , ?l ₃ → ?l ₅] [V _{RoR} ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]	[V _P ⁰ , ?f ₀ → ?p, ?f ₄ → ?f ₅] [V _P ⁰ , ?f ₄ → ?f ₅ , ?l ₆ → ?l ₅] [V _{RoR} ⁰ , ?r ₄ → ?r ₅ , ?l ₆ → ?l ₅]
	[V _P ⁰ , ?f ₀ → ?p, ?f ₁ → ?f ₅] [V _P ⁰ , ?f ₁ → ?f ₅ , ?l ₁ → ?l ₅] [V _P ⁰ , ?r ₁ → ?f ₅ , ?l ₃ → ?l ₅]	[V _P ⁰ , ?f ₀ → ?p, ?f ₄ → ?f ₅] [V _P ⁰ , ?f ₄ → ?f ₅ , ?l ₆ → ?l ₅]		
$V_P:$ CONSTRUCT{ ① ?f ₀ vfriend ?f ₁ . ② ?f ₁ vives ?l ₁ .} WHERE{ ?f ₀ name Eric, ?f ₀ friend ?f ₁ , ?f ₁ lives ?l ₁ .}	$V_{P \circ P}:$ CONSTRUCT{ ① ?f ₂ vfriend ?f ₄ . ② ?f ₄ vives ?l ₄ .} WHERE{ ?f ₂ name Eric, ?f ₂ friend ?f ₃ , ?f ₃ friend ?f ₄ , ?f ₄ lives ?l ₄ .}	$V_R:$ CONSTRUCT{ ① ?r ₀ vrelated ?r ₁ . ② ?r ₁ vives ?l ₃ .} WHERE{ ?r ₀ name Eric, ?r ₀ related ?r ₁ , ?r ₁ lives ?l ₃ .}	$V_{RoR}:$ CONSTRUCT{ ① ?r ₂ vrelated ?r ₄ . ② ?r ₄ vives ?l ₆ .} WHERE{ ?r ₂ name Eric, ?r ₂ related ?r ₃ , ?r ₃ related ?r ₄ , ?r ₄ lives ?l ₆ .}	

Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.



Optimization: pruning rewritings with empty results

- A solution: incremental rewriting.

